

BYTE

LISTINGS
SUPPLEMENT

January-June 1986

TABLE OF CONTENTS

January	3
February	73
March	201
April	217
May	317
June	343

INDEX

January

AIREAD	ME.....3
APPLGRF	BAS....5
CYCLFRAC	BAS....8
EDITSORT	MD2...13
FRAMES	PRO....9
GCD	BAS....7
LCM	BAS....7
LISTING	TXT...12
LISTING2	PRO....3
LISTING3	PRO....5
MACGRAF	BAS...12
PCGRAF	BAS...17
PROLOG	DOC...63
QUIKSORT	MD2...18
READSIM2	ME....16
SIMPL2	TXT...19

February

ANALYZER	ONE....76
ANALYZER	TWO....77
B10H14	DAT...133
B5H9	DAT...133
BADFILE	C.....120
BENZENE	DAT...135
BTRANS	PAS....73
C7H7O2N	DAT...134
CALC	PAS....73
CLOCKNET94
CLOCKPLT94
COLOR3D	BAS...131
CRC6H6	DAT...134
CRDIBENZ	DAT...135
DATAGEN	BAS...136
DVORAK	TXT...136
DVORAK	BAS...115
FARRELL	LBR...130
FARRELL	LIB...131

FIB	...123
FLOAT	PAS....74
FOO	NET....94
FOO	PLT....95
GOBBLER	ONE....78
GOBBLER	TWO....79
GRINDEF	...123
LEVIEN	LBR...122
LINETEST	PAS....73
LIST1	TXT...114
LIST2	TXT...114
LIST3	TXT...114
LIST4	TXT...115
LIST5	TXT...115
NACL	DAT...135
NATLANG	LBR....93
PARSE	PLT....96
PARSE	NET....95
PATTERNS	DAT...133
PUZZLE	PAS...118
QSORT	PAS....74
README	VSD...122
README3D	DOC...137
READNAT	ME....93
READSIM3	ME....92
SA	LSP....96
SA12	LSP...102
SCANPOEM	DOC...82
SCANPOEM	PAS....85
SIEVE	PAS...74
SMALLVSD	...123
ST	DOC...110
STRAMDSK	C.....107
TANKARD	LBR...76
TEXTBOX	DOC...82
TRANS	PAS....73
XLISPVSD	...127

March

DIOPHANT	BAS...214
EXPLORER	TXT...208
LISTING1	386...205
LISTING2	386...206
LISTING3	386...205
MFILECPY	MOD...204
MSCREEN	MOD...201
MSIEVE	MOD...201
MTIMEF	MOD...202
MTIMEI	MOD...203

April

ATSPGM	FOR...252
CHBEVL	C.....249
CHEBY	C.....250
DRAGON	DOC...313
EPSILON	FOR...234
EXAMPLES	DOC...235
LISTING4	PAS...247
MANUAL	DOC...255
MATINV	BAS...314
ONEBODY	BAS...239
ORBIT	FOR...229
PAINLEVE	FOR...237
PREDICT	FOR...225
REACT	FOR...230
REMES	C.....240
RKCONST	FOR...231
RUNKUT	DOC...217
RUNKUT	FOR...227
SIMQ	C.....245
THERM	FOR...235

May

HUFREAD	ME....334
MAINPROG	BAS...317
OPTICDB	C.....335
PELL	BAS...340
ROUTINE	SUB...341

June

FRACTAL	LIB...349
HILBERT	BAS...348
INDEXBPP	LST...343
INDEXBPP	PAS...364
LIST1	TXT...362
LIST2	TXT...364
MACVIEW	PAS...395
MIDI	ARC...352
MIDI111	C.....372
RXINT11	A.....390

BYTE

SENIOR VICE PRESIDENT/PUBLISHER
HARRY L. BROWN
EDITOR IN CHIEF
PHILIP LEMMONS

BIX

MANAGING EDITOR, BYTE
FREDERIC S. LANGA

ASSISTANT MANAGING EDITOR

GLENN HARTWIG

CONSULTING EDITORS

STEVE CIARCIA

JERRY POURNELLE

EZRA SHAPIRO

BRUCE WEBSTER

SENIOR TECHNICAL EDITORS

JON R. EDWARDS, *Reviews*

G. MICHAEL VOSE, *Themes*

GREGG WILLIAMS, *Features*

TECHNICAL EDITORS

DENNIS ALLEN

RICHARD GREHAN

KEN SHELDON

GEORGE A. STEWART

JANE MORRILL TAZELAAR

TOM THOMPSON

CHARLES D. WESTON

EVA WHITE

STANLEY WSZOLA

ASSOCIATE TECHNICAL EDITORS

CURTIS FRANKLIN, JR., *Best of BIX*

MARGARET COOK GURNEY, *Book Reviews*

BRENDA McLAUGHLIN, *Applications Software Reviews*

San Francisco

COPY EDITORS

BUD SADLER, *Chief*

JEFF EDMONDS

FAITH HANSON

NANCY HAYES

CATHY KINGERY

PAULA NOONAN

WARREN WILLIAMSON

JUDY WINKLER

ASSISTANTS

PEGGY DUNHAM, *Office Manager*

MARTHA HICKS

L. RYAN McCOMBS

CHAD MITCHELL

JUNE N. SHELDON

NEWS AND TECHNOLOGY

GENE SMARTE, *Bureau Chief, Costa Mesa*

JONATHAN ERICKSON, *Senior Technical Editor, San Francisco*

RICH MALLOY, *Senior Technical Editor, New York*

CINDY KIDDOO, *Editorial Assistant, San Francisco*

ASSOCIATE NEWS EDITORS

DENNIS BARKER, *Microbytes*

CATHRYN BASKIN, *What's New*

ANNE FISCHER LENT, *What's New*

CONTRIBUTING EDITORS

JONATHAN AMSTERDAM, *programming projects*

MARK DAHMKE, *video, operating systems*

MARK HAAS, *at large*

RIK JADRNICKEK, *CAD, graphics, spreadsheets*

ROBERT T. KUROSAKA, *mathematical recreations*

PHIL LOPICCOLO, *computers in medicine*

ALASTAIR J. W. MAYER, *software*

ALAN R. MILLER, *languages and engineering*

DICK POUNTAIN, *U.K.*

ROGER POWELL, *computers and music*

WILLIAM M. RAIKE, *Japan*

PHILLIP ROBINSON, *semiconductors*

ART

NANCY RICE, *Art Director*

JOSEPH A. GALLAGHER, *Associate Art Director*

JAN MULLER, *Art Assistant*

ALAN EASTON, *Drafting*

PRODUCTION

DAVID R. ANDERSON, *Production Director*

DENISE CHARTRAND

MICHAEL J. LONSKY

VIRGINIA REARDON

DANA RICE

TYPOGRAPHY

SHERRY MCCARTHY, *Chief Typographer*

LEN LORETTE

DONNA SWEENEY

EXECUTIVE EDITOR, BIX
GEORGE BOND

SENIOR EDITOR

DAVID BETZ

ASSOCIATE EDITORS

TONY LOCKWOOD

DONNA OSGOOD, *San Francisco*

BIX GROUP MODERATORS

DAVID P. ALLEN, *Applications Programs*

FRANK BOOSMAN, *Artificial Intelligence*

LEROY CASTERLINE, *Other*

MARC F. GREENFIELD, *Programming Languages and Tools*

JIM HOWARD, *Graphics*

GARY KENDALL, *Operating Systems*

STEVE KRENEK, *Personal Computers*

BROCK MEEKS, *Telecommunications*

BARRY NANCE, *New Technology*

DONALD OSGOOD, *Personal Computers*

SUE ROSENBERG, *Other*

ION SWANSON, *Chips*

BUSINESS AND MARKETING

DOUG WEBSTER, *Director, (603) 924-9027*

PATRICIA BAUSUM, *Secretary*

BRIAN WARNOCK, *Customer Service*

DENISE A. GREENE, *Customer Service*

TAMMY BURGESS, *Customer Credit and Billing*

TECHNOLOGY

CLAYTON LISLE, *Director, Business Systems Technology, MHIS*

BILL GARRISON, *Business Systems Analyst*

JACK REILLY, *Business Systems Analyst*

LINDA WOLFF, *Senior Business Systems Analyst*



Officers of McGraw-Hill Information Systems Company: President: Richard B. Miller. Executive Vice Presidents: Frederick P. Jannott, Construction Information Group; Russell C. White, Computers and Communications Information Group; J. Thomas Ryan, Marketing and International. Senior Vice Presidents: Francis A. Shinal, Controller; Robert C. Violette, Manufacturing and Technology. Senior Vice Presidents and Publishers: Laurence Altman, Electronics Week; Harry L. Brown, BYTE; David J. McGrath, Construction Publications. Group Vice President: Peter B. McCuen, Communications Information. Vice President: Fred O. Jensen, Planning and Development.

Officers of McGraw-Hill, Inc.: Harold W. McGraw, Jr., Chairman; Joseph L. Dionne, President and Chief Executive Officer; Robert N. Landes, Executive Vice President and Secretary; Walter D. Serwatka, Executive Vice President and Chief Financial Officer; Shel F. Asen, Senior Vice President, Manufacturing; Robert J. Bahash, Senior Vice President, Finance and Manufacturing; Ralph R. Schulz, Senior Vice President, Editorial; George R. Elsinger, Vice President, Circulation; Ralph J. Webb, Vice President and Treasurer.

BYTE, and The Small Systems Journal are registered trademarks of McGraw-Hill Inc.

EDITORIAL AND BUSINESS OFFICE: One Phoenix Mill Lane, Peterborough, New Hampshire 03458. (603) 924-9281.

West Coast Offices: 425 Battery St., San Francisco, CA 94111. (415) 954-9718; 3001 Red Hill Ave., Building #1, Suite 222, Costa Mesa, CA 92626. (714) 557-6292. **New York Editorial Office:** 1221 Avenue of the Americas, New York, NY 10020. (212) 512-2000.

BYTEnet: (617) 861-9764 (set modem at 8-I-N or 7-2-E; 300 or 1200 baud).

BYTE (ISSN 0360-5280) is published monthly with one extra issue per year by McGraw-Hill Inc. Founder: James H. McGraw (1860-1948). Executive, editorial, circulation, and advertising offices: 70 Main St., Peterborough, NH 03458, phone (603) 924-9281. Office hours: Mon-Thur 8:30 AM - 4:30 PM, Friday 8:30 AM - 1:00 PM. Eastern Time. Address subscriptions to BYTE Subscriptions, POB 590, Martinville, NJ 08836. Postmaster: send address changes USPS Form 3579; undeliverable copies, and fulfillment questions to BYTE Subscriptions, POB 596, Martinsville, NJ 08836. Second-class postage paid at Peterborough, NH 03458 and additional mailing offices. Postage paid at Winnipeg, Manitoba. Registration number 9321. Subscriptions are \$21 for one year, \$38 for two years, and \$55 for three years in the USA and its possessions. In Canada and Mexico, \$23 for one year, \$42 for two years, and \$61 for three years. \$69 for one year air delivery to Europe, \$1,000 yen for one year air delivery to Japan, \$1,600 yen for one year surface delivery to Japan, \$37 surface delivery elsewhere. Air delivery to selected areas at additional rates upon request. Single copy price is \$3.50 in the USA and its possessions, \$4.25 in Canada and Mexico, \$4.50 in Europe, and \$5 elsewhere. Foreign subscriptions and sales should be remitted in United States funds drawn on a U.S. bank. Please allow six to eight weeks for delivery of first issue. Printed in the United States of America.

Address all editorial correspondence to the Editor, BYTE, POB 372, Hancock, NH 03449. Unacceptable manuscripts will be returned if accompanied by sufficient first-class postage. Not responsible for lost manuscripts or photos. Opinions expressed by the authors are not necessarily those of BYTE.

Copyright © 1986 by McGraw-Hill Inc. All rights reserved. Trademark registered in the United States Patent and Trademark Office. Where necessary, permission is granted by the copyright owner for libraries and others registered with the Copyright Clearance Center (CCC) to photocopy any article herein for the flat fee of \$1.50 per copy of the article or any part thereof. Correspondence and payment should be sent directly to the CCC, 29 Congress St., Salem, MA 01970. Specify ISSN 0360-5280/86 \$1.50. Copying done for other than personal or internal reference use without the permission of McGraw-Hill Inc. is prohibited. Requests for special permission or bulk orders should be addressed to the publisher. BYTE is available in microform from University Microfilms International, 300 North Zeeb Rd., Dept. PR, Ann Arbor, MI 48106 or 18 Bedford Row, Dept. PR, London WC1R 4EJ, England.

Subscription questions or problems should be addressed to: BYTE Subscriber Service, POB 328, Hancock, NH 03449.



January

airead.me

TEXT

"AI in Computer Vision" John L. Cuadrado and Clara Y. Cuadrado.
January, page 237. Also download frames.pro listing2.pro, and listing3.pro
NOTE: Runs under Pdprolog.exe which can be found in the FromBYTE85 file area.
PDProlog is an MS-DOS or PC-DOS program only.

The prolog listing FRAMES.PRO will compile and run under PDPROLOG. The major limitation of PDPROLOG in this context is that it does not support floating-point arithmetic. We have truncated the value of pi in the cylinder cross-sectional area routine to 3. An alternative approach would be to use, e.g., 314 and mentally divide each area and volume by 100 to obtain an answer to two decimal points. In any case, when you enter values from the keyboard for, e.g., the Radius, you must use integer values if you are running PDPROLOG.

To invoke the program, boot PDPROLOG and type:
consult('frames').
After the program has compiled, type:
frame_put(cylinder1,radius,2).
frame_put(cylinder1,height,10).
etc.

In order to see the affect of entering a value for the radius or height, you may either type:
listing(cylinder1).
(Don't forget the period at the end of commands).
Or, you may use the frame_get predicate:
frame_get(cylinder1,X,Y).
X and Y must be capital letters. Initial caps indicate variable names in Prolog.

To change values for cylinder1, you may use either
frame_remove(cylinder1,radius). [or
frame_remove(cylinder1,name-of-slot-to-be-deleted).] followed by
frame_put(ENTER NEW VALUE AS EXPLAINED FOR FRAME_PUT ABOVE),
or frame_replace(cylinder1,radius,4). or whatever value you want changed.
This form allows the new value to be entered in one step, but is otherwise equivalent to frame_remove folloed by frame_put.

To end a pdprolog session and return to the operating system level, enter "exitsys." (without the quotes, of course).

LISTING2.PRO AND LISTING3.PRO ARE NOT STAND-ALONE PROGRAMS. THEY CANNOT BE EXECUTED.

The current version of pdprolog, pdprolog 1.8 is included on this disk. Full documentation and other sample prolog programs are available for downloading from BYTEnet Listings (617)861-9764 or from BIX.

PDPROLOG.EXE IS A PC-DOS OR MS-DOS PROGRAM ONLY.

listing2.pro

TEXT

"AI in Computer Vision." See airead.me.

window_type1

ako
value : window
panes
value : 12
style

value : sash

window_type2

ako
value : window
panes

(continued)

```

        value : 24
    style
        value : sash
window_type3
    ako
        value : window
    panes
        value : 3
    style
        value : picture
window_type4
    ako
        value : window
    panes
        value : 3
    style
        value : sash
window_type5
    ako
        value : window
    panes
        value : 2
    style
        value : sash
window
    ako
        value : thing
    area
        if_needed : window_area
window_area(Window,Area) :-
    fget(Window,height,Height),
    fget(Window,width,Width),
    Area is Height * Width,
    freplace(Window,area,Area).
door_type1
    ako
        value : door
    panels
        value : 4
    symmetry
        value : yes
    doorway
        value : [columns,fan_light]
door_type2
    ako
        value : door
    panels
        value : 6
    symmetry
        value : yes
    doorway
        value : [columns, portico,
                side_windows]
door_type3
    ako
        value : door
    panels
        value : 0
    symmetry
        value : no
    doorway
        value : []
door
    ako
        value : thing
    area
        if_needed : door_area

```

```

door_area(Door,Area) :-
    fget(Door,height,Height),
    fget(Window,width,Width),
    Area is Height * Width,
    freplace(Door,area,Area).

```

```

siding_type1
    ako
        value : siding
    material
        value : clapboard
    width
        value : narrow
    cornerboard
        value : yes

```

```

siding_type2
    ako
        value : siding
    material
        value : aluminum
    width
        value : wide
    cornerboard
        value : no

```

```

siding
    ako
        value : thing

```

```

house_type1
    ako
        value : house
    stories
        value : 3
    siding
        value : siding_type1
    roof
        value : gable
    window1
        optional : yes
        xposition : 2
        yposition : 3
        type : window_type2
    window2
        xposition : 1
        yposition : 2
        type : window_type2
    window3
        xposition : 3
        yposition : 2
        type : window_type2

```

```

proto_house
    ako
        value : house_type1
    window4
        xposition : 1
        yposition : 1
        type : window_type2
    door
        xposition : 3
        yposition : 1
        type : door_type1

```

```

proto_house_mirror_image
    ako
        value : house_type1
    window4
        xposition : 3
        yposition : 1
        type : window_type2
    door
        xposition : 1
        yposition : 1
        type : door_type1

```


listing3.pro

TEXT

"AI in Computer Vision." See aired.me.

```

house17
  ako
    value : house
  stories
    value : 3
  siding
    value : siding_type2
  roof
    value : gable
  window1
    value : w7
  window2
    value : w12
  window3
    value : w17
  window4
    value : w23
  door
    value : door37

w7
  ako
    value : window_type4
  ipo
    value : house17 /* is_part_of */
  xposition
    value : 2
  yposition
    value : 3

w12
  ako
    value : window_type4
  ipo
    value : house17

xposition
  value : 1
yposition
  value : 2

w17
  ako
    value : window_type4
  ipo
    value : house17
  xposition
    value : 3
  yposition
    value : 2

w23
  ako
    value : window_type3
  ipo
    value : house17
  xposition
    value : 3
  yposition
    value : 1

door37
  ako
    value : door_type3
  ipo
    value : house17
  xposition
    value : 1
  yposition
    value : 1

```

applgraf.bas

TEXT

Programming Insight: "Easy 3-D Graphics," Henning Mittelbach.
January, page 153. Apple version.

```

10 REM *****
20 REM      APPLGRAF
30 REM      EASY 3-D GRAPHICS, BY
40 REM      HENNING MITTELBACH
50 REM      FOR PRIVATE,
60 REM      NON-COMMERCIAL USE ONLY
70 REM *****
90 DIM H(279)
100 X0 = 110
110 Y0 = 180
120 PHI = .5
130 PSI = .4
140 XL = 0
150 XR = 170
160 YL = 0
170 YR = 100
180 D = 5
190 REM * FUNCTION TO BE PLOTTED *
200 DEF FN Y(X) = SIN (Y / F) * (X - Y) * (X - Y) / 150
210 F = 10
240 REM *ABBREVIATIONS AND CUTTING THE TOP *
250 CF = COS (PHI):SF = SIN (PHI):CP = COS (PSI):SP = SIN (PSI)
260 H = Y0 - XR * SF - YR * SP - 2

```

(continued)

```

270 PRINT "DO YOU DESIRE CROSS-HATCHING? (Y/N):"
275 INPUT OPT$
278 CH = 1
280 IF OPT$ = "Y" OR OPT$ = "y" THEN CH = 2
300 PRINT "DO YOU WISH TO VIEW THE AXES? (Y/N):"
305 INPUT AX$: HGR2 : HCOLOR= 7
310 IF AX$ = "N" OR AX$ = "n" THEN 340
320 HPLLOT X0 + XL * CF, Y0 - XL * SF TO X0 + XR * CF, Y0 - XR * SF
330 HPLLOT X0 - YL * CP, Y0 - YL * SP TO X0 - YR * CP, Y0 - YR * SP
340 HPLLOT 0,0 TO 279,0 TO 279,189 TO 0,189 TO 0,0
400 FOR R = 1 TO CH
420 FOR I = 0 TO 279:H(I) = 189: NEXT I
430 ON R GOSUB 1000,2000
440 NEXT R
500 HCOLOR= 0
510 HPLLOT 0,0 TO 279,0 TO 279,189 TO 0,189 TO 0,0
520 REM PRINTER ROUTINE FOR C.I.TOH
521 REM 8510 A PRINTER FOLLOWS.
522 REM DELETE "REM" COMMANDS TO USE
530 REM INPUT "PRINTING? (Y/N) ";J$
540 REM IF J$ = "N" THEN 600
560 REM PR#1
580 REM PRINT CHR$ ($14)
590 POKE 1657,96: CALL 49661!
600 REM PR#0
700 END
1000 REM
1010 Y = YL
1020 FOR X = XL TO XR
1030 XB = INT (X0 + X * CF - Y * CP + .5)
1040 Z = FN Y(X): IF Z > H THEN Z = H
1050 YB = INT (Y0 - X * SF - Y * SP - Z + .5)
1060 IF YB < H(XB) THEN H(XB) = YB
1070 NEXT X
1100 FOR X = XL TO XR STEP D
1110 U = X0 + X * CF:V = Y0 - X * SF
1120 FOR Y = YL TO YR
1130 XB = INT (U - Y * CP + .5)
1140 Z = FN Y(X): IF Z > H THEN Z = H
1150 YB = INT (V - Y * SP - Z + .5)
1160 IF YB < H(XB) THEN H(XB) = YB
1170 NEXT Y
1200 FOR K = INT (U - YR * CP + .5) TO INT (U - YL * CP + .5) - 1
1210 HPLLOT K,H(K) TO K + 1,H(K + 1)
1220 NEXT K
1230 NEXT X
1240 RETURN
2000 REM
2010 X = XL
2020 FOR Y = YL TO YR
2030 XB = INT (X0 + X * CF - Y * CP + .5)
2040 Z = FN Y(X): IF Z > H THEN Z = H
2050 YB = INT (Y0 - X * SF - Y * SP - Z + .5)
2060 IF YB < H(XB) THEN H(XB) = YB
2070 NEXT Y
2100 FOR Y = YL TO YR STEP D
2110 U = X0 - Y * CP:V = Y0 - Y * SP
2120 FOR X = XL TO XR
2130 XB = INT (U + X * CF + .5)
2140 Z = FN Y(X): IF Z > H THEN Z = H
2150 YB = INT (V - X * SF - Z + .5)
2160 IF YB < H(XB) THEN H(XB) = YB
2170 NEXT X
2200 FOR K = INT (U + XL * CF + .5) TO INT (U + XR * CF) - 1
2210 HPLLOT K,H(K) TO K + 1,H(K + 1)
2220 NEXT K
2230 NEXT Y
2240 RETURN

```


gcd.bas

TEXT

Mathematical Recreations: "Euclid's Algorithm," by Robert T. Kurosaka.
January page 397. Also download lcm.bas and cyclfrac.bas.

```

310 '*****
320 '*  EUCLID'S ALGORITHM FOR GREATEST COMMON DIVISORS  *
330 '*                      BY ROBERT T. KUROSAKA        *
340 '*****
350 CLS
360 PRINT "This program calculates the greatest common divisor"
370 PRINT "of a positive fraction"
380 PRINT "and reduces the fraction to lowest terms."
390 PRINT :PRINT
400 INPUT "ENTER THE FRACTION'S NUMERATOR";NUM:NUM=ABS(NUM)
410 INPUT "ENTER THE FRACTION'S DENOMINATOR";DEN:DEN=ABS(DEN)
420 DIVISOR=NUM:DIVIDEND=DEN 'SAVE ORIGINAL VALUES FOR LATER DISPLAY
430 REM IF EITHER TERM IS NOT A WHOLE NUMBER, CLEAR THE DECIMAL.
440 IF DIVISOR<>INT(DIVISOR) OR DIVIDEND<>INT(DIVIDEND) THEN
    DIVISOR=DIVISOR*10:
    DIVIDEND=DIVIDEND*10:NUM=DIVISOR:DEN=DIVIDEND:GOTO 440
450 IF DIVISOR>DIVIDEND THEN SWAP DIVISOR, DIVIDEND
460 WHILE DIVISOR>0
470     QUOTIENT=INT(DIVIDEND/DIVISOR)
480     REMAINDER=DIVIDEND-DIVISOR*QUOTIENT
490     DIVIDEND=DIVISOR:DIVISOR=REMAINDER
500 WEND
510 PRINT :PRINT
520 PRINT "THE FRACTION ";NUM;"/";DEN;" HAS A G.C.D. OF ";DIVIDEND
530 IF DIVIDEND=1 THEN PRINT "THE FRACTION IS ALREADY IN LOWEST TERMS.":GOTO 560
540 PRINT "THE REDUCED FRACTION IS: ";NUM/DIVIDEND;" /";DEN/DIVIDEND;
550 IF DEN/DIVIDEND=1 THEN PRINT " = ";NUM/DIVIDEND
560 END

```

lcm.bas

TEXT

Mathematical Recreations: "Euclid's Algorithm." See
gcd.bas.

```

10 '*****
20 '*  LEAST COMMON MULTIPLE ALGORITHM                    *
30 '*                      BY ROBERT T. KUROSAKA        *
40 '*****
50 CLS
60 PRINT "This program calculates the least common multiple"
70 PRINT "of a set of positive integers."
80 PRINT
90 INPUT "HOW MANY INTEGERS ARE IN THE SET";TERMS:TERMS=INT(ABS(TERMS))
100 IF TERMS<2 THEN 400
110 REM NUMBER ARRAY HOLDS THE SET OF INTEGERS FOR WHICH THE LCM IS SOUGHT.
120 DIM NUMBER(TERMS)
130 PRINT :PRINT "ENTER THE INTEGERS ONE AT A TIME."
140 FOR I=1 TO TERMS
150     INPUT NUMBER(I)
160     NUMBER(I)=INT(ABS(NUMBER(I)))
170     IF NUMBER(I)=0 THEN PRINT "ILLEGAL ENTRY.":GOTO 150
180 NEXT I
190 REM BEGIN LCM PROCEDURE.
200 LCM=NUMBER(1) 'THE LCM OF A SINGLE NUMBER IS ITSELF.
210 FOR I=2 TO TERMS
220     REM FIND GCD OF ACTIVE ENTRY AND WHAT PRECEDED IT (GCD WILL BE STORED
        IN 'DIVIDEND' BECAUSE LINE 290 ASSIGNS LAST VALID DIVISOR TO
        DIVIDEND).
230     DIVISOR=NUMBER(I):DIVIDEND=LCM
240     REM LINES 250-300 ARE THE SAME AS 450-500 OF THE GCD ROUTINE.
250     IF DIVISOR>DIVIDEND THEN SWAP DIVISOR,DIVIDEND
260     WHILE DIVISOR>0
270         QUOTIENT=INT(DIVIDEND/DIVISOR)

```

(continued)


```

280      REMAINDER=DIVIDEND-DIVISOR*QUOTIENT
290      DIVIDEND=DIVISOR:DIVISOR=REMAINDER
300      WEND
310      LCM=NUMBER(I)*LCM/DIVIDEND
320      REM THE LAST LCM WILL BE LCM OF ALL THE ENTRIES.
330 NEXT I
340 PRINT :PRINT
350 PRINT "THE LEAST COMMON MULTIPLE OF";
360 FOR I=1 TO TERMS
370     PRINT NUMBER(I);
380 NEXT I
390 PRINT "IS";LCM
400 END

```

cyclfrac.bas

TEXT

Mathematical Recreations: "Euclid's Algorithm," Robert T. Kurosaka.
Listing 3, page 402.

```

10 '*****
20 '*   REPEATING DECIMAL TO FRACTION CONVERTING ROUTINE   *
30 '*   BY ROBERT T. KUROSAKA                               *
40 '*****
50 CLS
60 PRINT "This routine can be used with the greatest common denominator"
70 PRINT "program. Load the GCD program, then MERGE this routine into it."
80 PRINT "The MERGED program is designed to determine the reduced fractional"
90 PRINT "representation of a repeating decimal.":PRINT
100 PRINT "To ENTER a repeating decimal:":
    PRINT "Type the nonrepeating part, then a '_' before the cycle."
110 PRINT "For example, 1.2_345 is the proper entry for 1.2345345345..."
120 PRINT "The decimal should always precede the '_', i.e., .333... is"
    PRINT "entered": PRINT "as '._3'. Reversing the '.' and '_' will cause"
    PRINT "an error.":PRINT
130 INPUT "ENTER REPEATING DECIMAL";NUMBER$
140 REM NON-REPEATING PART OF NUMBER IS THAT PART UP TO "_". VAL OPERATOR
    IGNORES ALL NUMBERS AFTER A NON-NUMERICAL CHARACTER. THUS, IN
    1.2_345, VAL("1.2_345") WILL BE 1.2, ETC.
150 NONREPEATING.PART=ABS(VAL(NUMBER$))
160 REM DEFINE A MORE READABLE FUNCTION TO USE FOR THROWING THE LEFTMOST
    CHARACTER OF A STRING AWAY.
170 DEF FNDROP.LEFT$(A$)=RIGHT$(A$,LEN(A$)-1)
180 REM FIND DECIMAL POINT
190 WHILE LEFT$(NUMBER$,1) <> "."
200     NUMBER$=FNDROP.LEFT$(NUMBER$)
210 WEND
220 NUMBER$=FNDROP.LEFT$(NUMBER$)
230 REM FIND OUT HOW MANY DECIMAL PLACES THE REPEATING CYCLE IS OFFSET FROM
    THE DECIMAL POINT.
240 WHILE LEFT$(NUMBER$,1) <> "_"
250     OFFSET=OFFSET+1
260     NUMBER$=FNDROP.LEFT$(NUMBER$)
270 WEND
280 REM THROW AWAY REPEATING PORTION MARKER, "_"
290 NUMBER$=FNDROP.LEFT$(NUMBER$)
300 REM HOW MANY DECIMAL PLACES ARE IN THE CYCLE? SINCE THE REPEATING CYCLE
    IS EVALUATED AFTER THROWING AWAY THE DECIMAL POINT, MULTIPLY BY
    10^(TOTAL NUMBER OF PLACES TO THE RIGHT IT SHOULD BE SHIFTED).
310 CYCLE.LENGTH=LEN(NUMBER$)
320 REPEATING.CYCLE=VAL(NUMBER$)*10^(OFFSET+CYCLE.LENGTH)
330 REM NUMBER=NONREPEATING PART+REPEATING CYCLE. SINCE THE FIRST ITERATION
    OF THE CYCLE IS THE ONLY ONE THAT DOES NOT CANCEL ON SUBTRACTION, ONLY USE
    IT.
340 NUMBER=NONREPEATING.PART+REPEATING.CYCLE
350 REM "CLEARED.NUMBER IS THE VALUE OF THE SUBTRACTION THAT DOES AWAY WITH
    THE INFINITE CYCLE (STEP 3 IN THE BYTE ARTICLE ALGORITHM).
360 CLEARED.NUMBER=NUMBER*10^CYCLE.LENGTH-NONREPEATING.PART
370 REM NOW, ASSIGN THE VALUES OF THE NUMERATOR AND DENOMINATOR TO THE
    VARIABLE NAMES USED IN THE GCD ROUTINE.
380 NUM=CLEARED.NUMBER:DEN=10^CYCLE.LENGTH-1
390 REM I CONVERT NUM AND DEN TO STRINGS AND THEN BACK TO CLEAR THE GUARD
    DIGITS IN THE NUM AND DEN VARIABLES. SEE BYTE ARTICLE FOR DETAILS.

```



```

400 NUM$=STR$(NUM):DEN$=STR$(DEN):NUM=VAL(NUM$):DEN=VAL(DEN$)
410 PRINT "THE EQUIVALENT UNREDUCED FRACTION IS: ";NUM;"/";DEN

```

```
frames.pro
```

```
TEXT
```

```
"AI in Computer Vision." See aired.me for details.
```

```

/* get the Value of Slot in a given Frame */
frame_get(Frame,Slot,Value) :-
    fget(Frame,Frame,Slot,Value).

fget(Parameter_Frame,Frame,Slot,Value) :- /* check for a value Facet */
    fget(Frame,Slot,value,Value).
fget(Parameter_Frame,Frame,Slot,Value) :- /* does it have a default? */
    fget(Frame,Slot,default,Value).
fget(Parameter_Frame,Frame,Slot,Value) :- /* how about a demon? */
    fget(Frame,Slot,if_needed,Rule),
    F =.. [Rule,Parameter_Frame,Value],
    F.
fget(Parameter_Frame,Frame,Slot,Value) :- /* none of the above */
    fget(Frame,ako,value,Parent), /* so, move up the hierarchy */
    fget(Parameter_Frame,Parent,Slot,Value).

fget(Frame,Slot,Facet,Value) :- /* just grab the given Facet or fail */
    F =.. [Frame,Slot,Facet,Value],
    F.

/* put Value in Slot of a given Frame. If this Slot has an associated
   if_added demon, then grab it and execute it after installing the
   given Value.
*/
frame_put(Frame,Slot,Value) :-
    get_rule(Frame,Slot,if_added,Rule), /* must we do something extra */
    fput(Frame,Slot,value,Value),
    F =.. [Rule,Frame,Value],
    F.
frame_put(Frame,Slot,Value) :-
    fput(Frame,Slot,value,Value). /* just a simple fput will do */

fput(Frame,Slot,Facet,Value) :-
    F =.. [Frame,Slot,Facet,Value],
    assertz(F).

/* remove Slot from a given Frame. If the Slot has an associated
   if_removed demon, then grab the rule and execute it before ,
   removing the Slot.
*/
frame_remove(Frame,Slot) :-
    get_rule(Frame,Slot,if_removed,Rule), /* something extra to do */
    F =.. [Rule,Frame],
    F,
    remove(Frame,Slot).
frame_remove(Frame,Slot) :-
    remove(Frame,Slot). /* just a simple remove */

remove(Frame,Slot) :-
    F =.. [Frame,Slot,value,Value],
    retract(F).
remove(_,_). /* if Slot doesn't exist, then no harm done */

/* replace whatever is in Slot with Value. If the Slot has an associated
   if_replaced rule, then grab it and execute it after doing the
   replacement.

```

(continued)

```

*/
frame_replace(Frame,Slot,Value) :-
    get_rule(Frame,Slot,if_replaced,Rule), /* something extra to do */
    freplace(Frame,Slot,Value),
    F =.. [Rule,Frame],
    F.
frame_replace(Frame,Slot,Value) :-
    freplace(Frame,Slot,Value).          /* just a simple replace */

freplace(Frame,Slot,Value) :-
    fremove(Frame,Slot),
    frame_put(Frame,Slot,Value).

```

```

/* append Value to the list in Slot. If Slot has an associated
   if_appended rule, then grab it and execute it after appending
   the Value.

```

```

*/
frame_append(Frame,Slot,Value) :-
    get_rule(Frame,Slot,if_appended,Rule),
    fappend(Frame,Slot,Value),
    F =.. [Rule,Frame],
    F.
frame_append(Frame,Slot,Value) :-
    fappend(Frame,Slot,Value).

```

```

/* here we check to see if the slot already exists.
   If it does, then we just append the new Value to the old value list.
   If the Slot does not exist, then we create it and give it a value
   consisting of the list whose single element is Value.

```

```

*/
fappend(Frame,Slot,Value) :-
    fget(Frame,Slot,value,Old),
    (member(Value,Old)
    ;
    fremove(Frame,Slot),
    fput(Frame,Slot,value,[Value|Old])
    ).
fappend(Frame,Slot,Value) :-
    fput(Frame,Slot,value,[Value]).

```

```

/* this is a simple utility predicate used to travel up the frame
   hierarchy looking for an appropriate rule to grab.

```

```

*/
get_rule(Frame,Slot,Type,Rule) :-
    fget(Frame,Slot,Type,Rule).
get_rule(Frame,Slot,Type,Rule) :-
    fget(Frame,ako,value,Parent),
    get_rule(Parent,Slot,Type,Rule).

```

```

/* Example

```

```

frame representation:

```

```

cylinder
  ako
    value : thing
  height
    if_added : cylinder_height_add
    if_removed : cylinder_height_remove
  radius
    if_added : cylinder_radius_add
    if_removed : cylinder_radius_remove
  cross_section
    if_needed : cylinder_cross_section
  volume
    if_needed : cylinder_volume

```



```
cylinder1
    ako
        value : cylinder
```

comments: cylinder1 above is an instance of cylinder. When we use frame_put(cylinder1,radius,2), say, the system will install the number "2" as the value of cylinder1's radius and it will further compute cylinder1's cross sectional area and install it under the cross_section slot. Similar actions take place when we do a frame_put for cylinder1's height. Below is the Prolog code that implements all this. NOTE: PDPROLOG only supports integer arithmetic.

```
*/
```

```
cylinder(ako,value,geometric_object).
cylinder(height,if_added,cylinder_height_add).
cylinder(height,if_removed,cylinder_height_remove).
cylinder(radius,if_added,cylinder_radius_add).
cylinder(radius,if_removed,cylinder_radius_remove).
cylinder(cross_section,if_needed,cylinder_cross_section).
cylinder(volume,if_needed,cylinder_volume).
```

```
/* if we get the height, then we try to compute the cylinder's
   volume.
```

```
*/
cylinder_height_add(Cylinder,_) :-
    cylinder_volume(Cylinder,_).
cylinder_height_add(_,_).      /* if we can't do it,
                               e.g., the radius is unknown,
                               then no harm done */
```

```
/* if the height is removed, then the old volume is no
   longer valid
```

```
*/
cylinder_height_remove(Cylinder) :-
    frame_remove(Cylinder,volume).
```

```
/* if we get the radius, then we can compute the cylinder's
   cross sectional area
```

```
*/
cylinder_radius_add(Cylinder,_) :-
    cylinder_cross_section(Cylinder,_).
```

```
/* if the radius is removed, then the old cross sectional area
   is no longer valid
```

```
*/
cylinder_radius_remove(Cylinder) :-
    frame_remove(Cylinder,cross_section),
    frame_remove(Cylinder,volume).
```

```
/* PDPROLOG does not support floating-point arithmetic, so pi has been
   approximated as 3. If you are using a commercial prolog, change 3 to 3.1416
```

```
*/
cylinder_cross_section(Cylinder,Cross_Section) :-
    frame_get(Cylinder,radius,Radius),
    Cross_Section is 3*Radius*Radius,
    freplace(Cylinder,cross_section,Cross_Section).
```

```
cylinder_volume(Cylinder,Volume) :-
    frame_get(Cylinder,cross_section,Cross_Section),
    frame_get(Cylinder,height,Height),
    Volume is Height*Cross_Section,
    freplace(Cylinder,volume,Volume).
```

```
cylinder1(ako,value,cylinder).
```

listing.txt

TEXT

Ciarcia's Circuit Cellar: "Build an Analog-to-Digital Converter," Steve Ciarcia.
January, page 104.

```

10 CLEAR
20 REM READ AND DISPLAY A/D CHANNELS 0-7
30 REM SINGLE ENDED OR DIFFERENTIAL
40 REM -5 TO +5 VOLT INPUT
50 REM
60 REM
70 N=47104 : REM BOARD ADDRESS
80 REM STATUS BIT IS B5 - LOGIC 1 IS RESET
90 FOR A=0 TO 7 : REM DO ALL CHANNELS 0-7
100 GOSUB 160 : REM READ A CHANNEL
110 NEXT A : REM NEXT CHANNEL
120 PRINT CHR(18),CHR(27),"Y" : REM TERMITE - HOME AND CLEAR SCREEN
130 REM DISPLAY ARRAY HOLDING CHANNEL 0-7 READINGS
140 PRINT USING(###),A(0),A(1),A(2),A(3),A(4),A(5),A(6),A(7),"VOLTS"
150 GOTO 20 : REM DO IT AGAIN
160 XBY(N)=A + 16 : REM RESET A/D AND SET MUX CHANNEL
170 XBY(N)=A : REM CLEAR STATUS BIT TO READ DATA
180 D1=XBY(N) : D2=XBY(N) : REM READ 12 BITS AS 2 SUCCESSIVE WORDS
190 R=.0012207 : REM VOLTS PER COUNT
200 IF D1>240 THEN GOTO 230
210 A(A)=R*((D1*256)+D2) : REM SAVE POSITIVE READING IN ARRAY
220 RETURN
230 D1=255-D1 : D2=255-D2 : REM ADJUST D1 & D2 FOR 2'S COMPLEMENT
240 A(A)=-1*R*((D1*256)+D2) : REM SAVE NEGATIVE READING IN ARRAY
250 RETURN

```

macgraf.bas

TEXT

Programming Insight: "Easy 3-D Graphics," Henning Mittlebach.
January, page 153. Macintosh version.

```

10 ' *****
20 ' MacGraf - Easy 3-D Graphics on the Macintosh
30 ' by
40 ' Henning Mittelbach
50 ' Copyright 1985, for private, non-commercial
60 ' use only.
70 ' *****
80 CLS
90 DIM H (279)
100 X0=110
110 Y0=180
120 PHI=.5
130 PSI=.4
140 XL= 0
150 XR= 170
160 YL=0
170 YR=100
180 D=5
198 ' * FUNCTION TO BE PLOTTED *
199 '
200 DEF FN Y(X) = SIN (Y/F) * (X-Y) * (X-Y)/150
210 F=10
240 ' * ABBREVIATIONS AND CUTTING THE TOP *
250 CF= COS (PHI) : SF=SIN(PHI) : CP= COS (PSI) : SP= SIN(PSI)
260 H=Y0 - XR * SF -YR * SP - 2
270 INPUT "Do you desire cross-hatching? (Y/N):", opt$
280 IF opt$="Y" OR opt$="y" THEN ch=2 ELSE ch=1
300 INPUT "Do you wish to view the axes? (Y/N):", AX$
310 CLS : IF AX$="Y" OR ax$="y" THEN 320 ELSE 340
320 LINE (X0 + XL * CF, Y0 - XL * SF) -(X0 + XR * CF, Y0 - XR * SF)
330 LINE (X0 - YL * CP, Y0 - YL * SP) -(X0 - YR * CP, Y0 - YR * SP)
340 LINE (0,0) - (279,189),,B
350

```



```

398 ' R=1: Y-COORD. LINES
399 ' R=2: X-COORD. LINES
400 FOR R = 1 TO ch
410 ' * SETTING MASK ON LOWER BORDER OF WINDOW *
420 FOR I = 0 TO 279: H(I) = 189: NEXT I
430 ON R GOSUB 1000, 2000
440 NEXT R
499 ' * GRAPHIC IS FINISHED *
500 BEEP
510 LINE (0,0) - (279,189),,B
520 a$= INKEY$: IF a$= "" THEN 520
600 END
610 ' * END OF PROGRAM *
1000 ' * Y-COORD. LINES FOR X = CONST. *
1010 Y = YL: ' * FRONT MASK SETTING *
1020 FOR X = XL TO XR
1030 XB = INT (X0 + X * CF - Y * CP + .5)
1040 Z = FN Y(X) : IF Z > H THEN Z = H
1050 YB = INT (Y0 - X * SF - Y * SP - Z + .5)
1060 IF YB < H(XB) THEN H(XB) = YB
1070 NEXT X
1090 ' * ADAPTING THE MASK PER LINE *
1100 FOR X = XL TO XR STEP D
1110 U = X0 + X * CF : V = Y0 - X * SF
1120 FOR Y = YL TO YR
1130 XB = INT (U - Y * CP + .5)
1140 Z = FN Y(X) : IF Z > H THEN Z = H
1150 YB = INT (V - Y * SP - Z + .5)
1160 IF YB < H(XB) THEN H(XB) = YB
1170 NEXT Y
1190 ' * PLOTTING THE MASK *
1200 FOR K = INT (U - YR * CP + .5) TO INT (U - YL * CP + .5) - 1
1210 LINE (K,H(K)) - (K + 1,H(K+1))
1220 NEXT K
1230 NEXT X
1240 RETURN
2000 ' * X-COORD. LINES FOR Y = CONST. *
2010 X = XL: ' * FRONT MASK SETTING *
2020 FOR Y = YL TO YR
2030 XB = INT (X0 + X * CF - Y * CP + .5)
2040 Z = FN Y(X) : IF Z > H THEN Z = H
2050 YB = INT (Y0 - X * SF - Y * SP - Z + .5)
2060 IF YB < H(XB) THEN H(XB) = YB
2070 NEXT Y
2090 ' * ADAPTING THE MASK PER LINE *
2100 FOR Y = YL TO YR STEP D
2110 U = X0 - Y * CP : V = Y0 - Y * SP
2120 FOR X = XL TO XR
2130 XB = INT (U + X * CF + .5)
2140 Z = FN Y(X) : IF Z > H THEN Z = H
2150 YB = INT (V - X * SF - Z + .5)
2160 IF YB < H(XB) THEN H(XB) = YB
2170 NEXT X
2190 ' * PLOTTING THE MASK *
2200 FOR K = INT (U + XL * CF + .5) TO INT (U + XR * CF) - 1
2210 LINE (K,H(K)) - (K + 1,H(K+1))
2220 NEXT K
2230 NEXT Y
2240 RETURN

```

editsort.md2

TEXT

"Creating Reusable Modules," Namir Clement Shammas.
January, page 145. Also download quiksort.md2.

EDITSORT.MD2

MODULE EditSort;

```

(* ----- *)
(* This module is the capsule editor for the procedure QuickSort. *)

```

(continued)

```

(*) This editor will perform the following:
(*)
(*) (1) Customize the procedure name.
(*) (2) Customize the Record type declaration.
(*) (3) Customize the keys for sorting.
(*) -----
(*)

```

```

FROM Strlib1 IMPORT StringIs, StringAdd, StringRemove, StringReplace,
    ShowString, StringLeft, InputString, Len,
    StringPos, eos;
FROM FileSystem IMPORT File, Response, Lookup, Close, ReadChar,
    WriteChar;
FROM InOut IMPORT ReadCard, WriteCard;
FROM Terminal IMPORT WriteLn;

```

```

CONST MAXKEY = 10;
    MAXSTRING = 80;
    EOL = 36C ;

```

```

TYPE String = ARRAY [1..MAXSTRING] OF CHAR;

```

```

VAR i, j, k, n : CARDINAL;
    ch : CHAR;
    Line, Str, Sortname, Recname, YourFile, Fldname : String;
    Subkey : ARRAY [1..MAXKEY] OF String;
    f1, f2 : File;

```

```

PROCEDURE GetLine;
(*) Procedure to read the next text line from QWKSORT.CAP file.
(*) Insert an End Of String (eos) if string is not full.
(*)

```

```

BEGIN
    i := 0;
    REPEAT
        ReadChar(f1,ch);
        INC(i);
        Line[i] := ch
    UNTIL ch = CHAR(EOL) ;
    IF i < MAXSTRING THEN Line[i + 1] := eos END;
END GetLine;

```

```

PROCEDURE PutLine;
(*) Procedure to write a text line in the user specified output
(*) file. If the line is generated by this program, append an
(*) End-Of-Line (EOL) character to the text line.
(*)

```

```

BEGIN
    i := Len(Line);
    IF i>0 THEN
        FOR j := 1 TO i DO
            ch := Line[j];
            WriteChar(f2,ch)
        END;
        IF ch <> CHAR(EOL) THEN
            WriteChar(f2,CHAR(EOL)) END;
        END;
END PutLine;

```

```

PROCEDURE ScanSkip(Match : ARRAY OF CHAR);
(*) Procedure will read a text line from file QWKSORT.CAP and
(*) attempt to locate string 'Match' in it. If no match is
(*) found, the line is written to output text file.
(*)

```

```

VAR Pos : CARDINAL;

```

```

BEGIN
    Pos := 0;
    WHILE Pos = 0 DO
        GetLine;
        Pos := StringPos(Line,Match,1);
        IF Pos = 0 THEN PutLine END;
    END;
END ScanSkip;

```

```

PROCEDURE OneSort;
(*) Procedure to customize the dummy key field
(*)

```



```

BEGIN
  (* Edit record type. *)
  ScanSkip("Item");
  StringReplace(Line,"Item",Recname); PutLine;
  (* Enter sort key and edit the 'dummy' key. *)
  ShowString("Enter fieldname ? "); InputString(Fldname);
  FOR k := 1 TO 2 DO
    ScanSkip("key"); StringReplace(Line,"key",Fldname);
    PutLine
  END;
END OneSort;

PROCEDURE MultiSort;
(* Procedure to establish multikey sorting. *)

BEGIN
  (* Enter the number of sort fields and their names. *)
  ShowString("Enter number of fields used ? ");
  ReadCard(n);
  WriteLn;
  FOR k := 1 TO n DO
    ShowString("Enter name for subkey # ");
    WriteCard(k,2);
    ShowString(" "); InputString(Subkey[k]);
    WriteLn
  END;
  (* Edit the arguments in the procedure call, changing them *)
  (* from arrays of character to the user specified record type. *)
  ScanSkip("S1, S2 :");
  Stringls(Str,"R1, R2 : ");
  StringAdd(Str,Recname); StringReplace(Line,"S1, S2 :
    ARRAY OF CHAR",Str);
  PutLine;
  ScanSkip("i : CARDINAL;"); PutLine;
  (* Insert the declaration for the strings used in the *)
  (* comparison. *)
  Stringls(Line," S1, S2 : ARRAY [1.YourMaxString] OF CHAR;");
  PutLine;
  ScanSkip("i := 0"); PutLine;
  (* Build the text line that represents the code for the *)
  (* build-up of the multifield sort string. *)
  Stringls(Line,"Stringls(S1,R1."); StringAdd(Line,Subkey[1]);
  StringAdd(Line,") ; Stringls(S2,R2.");
  StringAdd(Line,Subkey[1]);
  StringAdd(Line,") ;"); PutLine;
  IF n > 1 THEN
    FOR k := 2 TO n DO
      Stringls(Line," StringAdd(S1,R1.");
      StringAdd(Line,Subkey[k]);
      StringAdd(Line,") ; StringAdd(S2,R2.");
      StringAdd(Line,Subkey[k]);
      StringAdd(Line,") ;"); PutLine
    END;
  END;

  (* Edit record type for the locally declared records. *)
  ScanSkip("Item");
  StringReplace(Line,"Item",Recname); PutLine;
  (* Edit the call to the Compare procedure. *)
  FOR k := 1 TO 2 DO
    ScanSkip(".key") ; StringRemove(Line,".key"); PutLine
  END;
END MultiSort;

BEGIN (* Main module *)
  ShowString("Enter the output filename ? ");
  InputString(YourFile); WriteLn;
  Lookup(f1,"c:qwksort.cap","FALSE");
  Lookup(f2,YourFile,TRUE);
  (* Check if both files are opened correctly. *)
  IF (f1.res = done) AND (f2.res = done) THEN
    ShowString("Enter new procedure name ? ");
    InputString(Sortname);
    WriteLn;
    GetLine ; StringReplace(Line,"QuickSort","Sortname");
  
```

(continued)

```

ShowString("Enter record type name ? ");
InputString(Recname);
WriteLn;
StringReplace(Line,"Item",Recname); PutLine;
GetLine ; GetLine; (* Skip the two comment lines *)
ShowString("Is the sort based on one field ? ");
InputString(Str); WriteLn;
StringLeft(Str,Str,1); (* Extract the leftmost character *)
IF CAP(Str[1]) = "Y" THEN OneSort ELSE MultiSort
END;
ScanSkip("QuickSort") ; StringReplace(Line,"QuickSort",
Sortname);
PutLine;
ELSE
ShowString("Error in locating file QWKSORT.CAP")
END;
IF (f1.res = done) THEN Close(f1) END;
IF (f2.res = done) THEN Close(f2) END;
END EditSort.

```

readsim2.me

TEXT
"A SIMPL Compiler, Part 2: Procedures and Functions." See simpl2.txt.

ZRM700 Guide to Part 2 of the SIMPL Compiler

This code accompanies part 2 of "A SIMPL Compiler." It includes Modula-2 source code for the entire SIMPL Compiler. You will need the Monitor program from Building a Computer in Software (October '85) and the VM2 Assembler (November '85). There are 12 modules to the compiler:

```

CodeGen
CodeWrite
Compiler (MOD file only)
ExprParser
LexAn
Node
Parser
Routines
Symbol
SymbolTable
Token
TypeChecker

```

[Editor's note: The above files were combined into one file when they were put on the bulletin board. You will need to break each module into a separate file to compile them.]

You will also need to compile the following utility modules along with the 12 above. These modules can be found with those downloaded with the VM2 Monitor:

```

CharStuff
LexAnStuff
MyTerminal
StringStuff

```

The programs were developed using MacModula-2 for the Macintosh, but conversion to other Modula-2 systems should be straightforward. I would appreciate hearing about any conversion difficulties or bugs. You can reach me on BIX as "jba" or by U.S. mail at 1643 Cambridge St. #34, Cambridge, MA 02138. Happy Compiling!

Jonathan Amsterdam

pcgraf.bas

TEXT

Programming Insight: "Easy 3-D Graphics," Henning Mittlebach.
January, page 153. IBM version.

```

10 ' *****
20 ' * PCGRAF - Easy 3-D Graphics on the IBM PC *
30 ' * by
40 ' * Henning Mittlebach
50 ' * Copyright 1985, for private non-commercial
60 ' * use only.
70 ' *****
80 CLS: SCREEN 1
90 DIM H (279)
100 X0=110
110 Y0=180
120 PHI=.5
130 PSI=.4
140 XL= 0
150 XR= 170
160 YL=0
170 YR=100
180 D=5
198 ' * FUNCTION TO BE PLOTTED *
199 '
200 DEF FN Y(X) = SIN (Y/F) * (X-Y) * (X-Y)/150
210 F=10
240 ' * ABBREVIATIONS AND CUTTING THE TOP *
250 CF= COS (PHI) : SF=SIN(PHI) : CP= COS (PSI) : SP= SIN(PSI)
260 H=Y0 - XR * SF -YR * SP - 2
270 INPUT "Do you desire cross-hatching? (Y/N):",OPT$
280 IF OPT$="Y" OR OPT$="y" THEN CH=2 ELSE CH=1
300 INPUT "Do you wish to view the axes? (Y/N):",AX$
310 CLS: IF AX$ = "Y" OR AX$ = "y" THEN 320 ELSE 340
320 LINE (X0 + XL * CF, Y0 - XL * SF) -(X0 + XR * CF, Y0 - XR * SF)
330 LINE (X0 - YL * CP, Y0 - YL * SP) -(X0 - YR * CP, Y0 - YR * SP)
340 LINE (0,0) - (279,189),,B
350 '
398 ' R=1: Y-COORD. LINES
399 ' R=2: X-COORD. LINES
400 FOR R = 1 TO CH
410 ' * SETTING MASK ON LOWER BORDER OF WINDOW *
420 FOR I = 0 TO 279: H(I) = 189: NEXT I
430 ON R GOSUB 1000, 2000
440 NEXT R
499 ' * GRAPHIC IS FINISHED *
500 BEEP
510 LINE (0,0) - (279,189),,B
520 A$=INKEY$: IF A$="" THEN 520
600 END
610 ' * END OF PROGRAM *
1000 ' * Y-COORD. LINES FOR X = CONST. *
1010 Y = YL: ' * FRONT MASK SETTING *
1020 FOR X = XL TO XR
1030 XB = INT (X0 + X * CF - Y * CP + .5)
1040 Z = FN Y(X) : IF Z > H THEN Z = H
1050 YB = INT (Y0 - X * SF - Y * SP - Z + .5)
1060 IF YB < H(XB) THEN H(XB) = YB
1070 NEXT X
1090 ' * ADAPTING THE MASK PER LINE *
1100 FOR X = XL TO XR STEP D
1110 U = X0 + X * CF : V = Y0 - X * SF
1120 FOR Y = YL TO YR
1130 XB = INT (U - Y * CP + .5)
1140 Z = FN Y(X) : IF Z > H THEN Z = H
1150 YB = INT (V - Y * SP - Z + .5)
1160 IF YB < H(XB) THEN H(XB) = YB
1170 NEXT Y
1190 ' * PLOTTING THE MASK *
1200 FOR K = INT (U - YR * CP + .5) TO INT (U - YL * CP + .5) - 1
1210 LINE (K,H(K)) - (K + 1,H(K+1))

```

(continued)

```

1220 NEXT K
1230 NEXT X
1240 RETURN
2000 ' * X-COORD. LINES FOR Y = CONST. *
2010 X = XL: ' * FRONT MASK SETTING *
2020 FOR Y = YL TO YR
2030 XB = INT (X0 + X * CF - Y * CP + .5)
2040 Z = FN Y(X) : IF Z > H THEN Z = H
2050 YB = INT (Y0 - X * SF - Y * SP - Z + .5)
2060 IF YB < H(XB) THEN H(XB) = YB
2070 NEXT Y
2090 ' * ADAPTING THE MASK PER LINE *
2100 FOR Y = YL TO YR STEP D
2110 U = X0 - Y * CP : V = Y0 - Y * SP
2120 FOR X = XL TO XR
2130 XB = INT (U + X * CF + .5)
2140 Z = FN Y(X) : IF Z > H THEN Z = H
2150 YB = INT (V - X * SF - Z + .5)
2160 IF YB < H(XB) THEN H(XB) = YB
2170 NEXT X
2190 ' * PLOTTING THE MASK *
2200 FOR K = INT (U + XL * CF + .5) TO INT (U + XR * CF) - 1
2210 LINE (K,H(K)) - (K + 1,H(K+1))
2220 NEXT K
2230 NEXT Y
2240 RETURN

```

quicksort.md2

TEXT
 "Creating Reusable Modules." See editsort.md2.

QUIKSORT.MD2

(* This module should be saved under the name QWKSORT.CAP. *)

```

PROCEDURE QuickSort( A : ARRAY OF Item ; N : CARDINAL );
(* Capsule QuickSort: A skeleton procedure for using the *)
(* quicksort algorithm. See reference 3. *)

```

```

PROCEDURE Compare ( S1, S2 : ARRAY OF CHAR): BOOLEAN;
(* Compare two strings of the same maximum lengths. *)

```

CONST eos = 0C; (* End-Of-String *)

VAR Less, Stop : BOOLEAN;
 i : CARDINAL;

BEGIN

```

  Less := FALSE;
  Stop := FALSE;
  i := 0;
  WHILE (i <= HIGH(S1)) AND (Less = FALSE) AND (Stop = FALSE) DO
    IF (S1[i] <> eos) AND (S2[i] <> eos)
      THEN (* Proceed in comparison *)
        IF (S1[i] < S2[i]) THEN Less := TRUE ELSE INC(i) END;
        ELSE Stop := TRUE (* Reached the end of string *)
      END;
    END;
  RETURN Less;
END Compare;

```

PROCEDURE Sort(L, R : CARDINAL);

VAR i, j : CARDINAL;
 X, W : Item;

BEGIN

```

  X := A[(L + R) DIV 2];
  REPEAT
    WHILE Compare(A[i].key,X.key) DO INC(i) END;
    WHILE Compare(X.key,A[i].key) DO DEC(j) END;
    IF i <= j THEN

```



```

        W = A[i] := A[i] ; A[i] := A[j] ; A[j] := W ;
        INC(i) ; DEC(j)
    END;
UNTIL i > j ;
IF L < j THEN Sort(L,j) END;
IF i < R THEN Sort(i,R) END;
END Sort;

BEGIN

    Sort(1,N)

END QuickSort;

```

simpl2.txt

TEXT
 Programming Project: "A SIMPL Compiler, Part 2: Procedures and Functions," Jonathan Amsterdam.
 January, page 130. Code for the revised SIMPL compiler. Also download
 readsim2.me.

E+++++++
 Editor's note: Break each of the files into a separate file. Be sure and
 delete the notes surrounded by plus signs.

Start CodeGen.DEF
 ++++++++

DEFINITION MODULE CodeGen;

(* This module generates code from parse trees. *)

FROM Node IMPORT node;
 FROM Symbol IMPORT symbol;

EXPORT QUALIFIED genBlock, genGlobal, genLocals;

PROCEDURE genBlock(n:node);
 (* Generate code for a block of statements. *)

PROCEDURE genGlobal(s:symbol);
 (* Generate code for a global variable. *)

PROCEDURE genLocals(routine:symbol);
 (* Generate code to set up the stack for local variables. *)

END CodeGen.

+++++++

Start CodeGen.MOD

+++++++

IMPLEMENTATION MODULE CodeGen;

(* Code Generator for the SIMPL compiler. *)

IMPORT MyTerminal;

FROM InOut IMPORT WriteString, WriteLn;

FROM Node IMPORT node, nodeClass, NodeClass, nodeEmpty, nodeFirst, nodeRest,
 nodeTest, nodeThen, nodeElse, nodeStmts, nodeRHS, nodeLHS, nodeArgs,
 nodeRoutine, nodeExpr, nodeArg, nodeLeftArg, nodeRightArg, nodeOp,
 nodeSymbol, nodeInt, nodeBool, nodeNumFormals, nodeChar,
 nodeType, freeNode;

FROM CodeWrite IMPORT writeLabel, writeStringLabel, writeStringBranch,
 writeCondBranch, writeBranch, writePop, writeCall, writeChar,
 writeWriteInt, writeInt, writeReadInt, writeReturn, writeFReturn,
 writeOp, writeBool, writeSymbol, writeWriteChar, writeReadChar;

FROM Token IMPORT tokenClass, isRelation, stringType, typeType;

FROM LexAn IMPORT errorFlag;

FROM Symbol IMPORT symbol, symbolLexLevel, symbolString, numLocals;

(*** label generation ***)

(* The code generator needs a supply of unique label names. *)

(continued)

January

```
MODULE LabelGenerator;
EXPORT newLabel, label;

TYPE label = CARDINAL;

VAR labelCount: CARDINAL;

PROCEDURE newLabel(): label;
BEGIN
    INC(labelCount);
    RETURN label(labelCount);
END newLabel;

BEGIN
    labelCount := 0;
END LabelGenerator;

PROCEDURE genBlock(n: node);
(* This is the interface to generating statements. We don't waste our time
   doing generation if there has been an error. Also, it's possible for this
   routine to get an empty node (legally); in that case, we do nothing. *)
BEGIN
    IF (NOT errorFlag) AND (NOT nodeEmpty(n)) THEN
        IF nodeClass(n) <> nList THEN
            MyTerminal.fatal('genBlock: node class must be nList');
        ELSE
            genStmts(n);
            freeNode(n);
        END;
    END;
END genBlock;

PROCEDURE genGlobal(s: symbol);
(* Output the global symbol as a label. Initialize integers to 0, booleans
   to FALSE (which is also zero), chars to NUL (which is again zero). *)
VAR name: stringType;
BEGIN
    symbolString(s, name);
    IF symbolLexLevel(s) = 0 THEN
        writeStringLabel(name);
        WriteString(" 0");
        WriteLn;
    ELSE
        MyTerminal.WriteString("genGlobal: not a global: ");
        MyTerminal.fatal(name);
    END;
END genGlobal;

PROCEDURE genLocals(routine: symbol);
(* put locals on stack, initialized to 0 (or FALSE, or NUL) *)
VAR i: CARDINAL;
BEGIN
    FOR i := 1 TO numLocals(routine) DO
        writeInt(0);
    END;
END genLocals;

PROCEDURE genStmts(n: node);
(* Generate a list of statements, if the node isn't empty. *)
BEGIN
    IF NOT nodeEmpty(n) THEN
        genStmt(nodeFirst(n));
        genStmts(nodeRest(n));
    END;
END genStmts;

(***) Statements (***)

PROCEDURE genStmt(n: node);
BEGIN
    CASE nodeClass(n) OF
        nIf: genIfStmt(n);
        nWhile: genWhileStmt(n);
        nAssignment: genAssignStmt(n);
        nCall: genCallStmt(n);
    END;
END;
```



```

    nWrite: genWriteStmt(n);
    nRead: genReadStmt(n);
    nReturn: genReturnStmt(n);
ELSE
    MyTerminal.fatal("genStmt: unknown statement type");
END;
END genStmt;

PROCEDURE genIfStmt(n:node);
VAR label1, label2:label;
BEGIN
    label1 := newLabel();
    genExpr(nodeTest(n));
    writeCondBranch(Equal, label1);
    (* generate test *)
    (* branch to else part if test false *)
    *)
    genBlock(nodeThen(n));
    IF nodeEmpty(nodeElse(n)) THEN
        writeLabel(label1);
    ELSE
        label2 := newLabel();
        writeBranch(label2);
        writeLabel(label1);
        genBlock(nodeElse(n));
        writeLabel(label2);
    (* generate then part *)
    (* no else part *)
    (* branch around els part *)
    (* label for else part *)
    (* generate else part *)
    (* final label *)
    END;
END genIfStmt;

PROCEDURE genWhileStmt(n:node);
VAR testLabel, endLabel:label;
BEGIN
    testLabel := newLabel();
    endLabel := newLabel();
    writeLabel(testLabel);
    genExpr(nodeTest(n));
    writeCondBranch(Equal, endLabel);
    genBlock(nodeStmts(n));
    writeBranch(testLabel);
    writeLabel(endLabel);
    (* label for top of loop *)
    (* generate test *)
    (* if false, branch to end of loop *)
    (* generate loop body *)
    (* branch back to test *)
    (* end label *)
END genWhileStmt;

PROCEDURE genAssignStmt(n:node);
BEGIN
    genExpr(nodeRHS(n));
    writePop(nodeLHS(n));
    (* generate the expression *)
    (* pop the result into the variable *)
END genAssignStmt;

PROCEDURE genCallStmt(n:node);
BEGIN
    genExprList(nodeArgs(n));
    writeCall(nodeRoutine(n));
    (* generate the arguments *)
    (* generate a call instruction *)
END genCallStmt;

PROCEDURE genWriteStmt(n:node);
(* Generate code to write the arguments to the screen. WRITE can take any
   number of arguments. *)
VAR arglist:node;
BEGIN
    arglist := nodeArgs(n);
    WHILE NOT nodeEmpty(arglist) DO
        genExpr(nodeFirst(arglist));
        CASE nodeType(nodeFirst(arglist)) OF
            tInteger: writeWriteInt;
            | tChar: writeWriteChar;
        ELSE
            MyTerminal.fatal("genWriteStmt: illegal type");
        END;
        arglist := nodeRest(arglist);
    END;
END genWriteStmt;

PROCEDURE genReadStmt(n:node);
(* Generate code to read from the terminal. READ can take any number of
   arguments. *)
VAR arglist:node;

```

(continued)

```

BEGIN
  arglist := nodeArgs(n);
  WHILE NOT nodeEmpty(arglist) DO
    CASE nodeType(nodeFirst(arglist)) OF
      tInteger: writeReadInt;
      | tChar:   writeReadChar;
    ELSE
      MyTerminal.fatal("genReadStmt: illegal type");
    END;
    writePop(nodeSymbol(nodeFirst(arglist)));
    arglist := nodeRest(arglist);
  END;
END genReadStmt;

PROCEDURE genReturnStmt(n:node);
BEGIN
  IF nodeEmpty(nodeExpr(n)) THEN (* a procedure return *)
    writeReturn(nodeNumFormals(n));
  ELSE (* a function return *)
    genExpr(nodeExpr(n));
    writeFReturn(nodeNumFormals(n));
  END;
END genReturnStmt;

      (** expressions **)

PROCEDURE genExprList(n:node);
VAR el:node;
BEGIN
  el := n;
  WHILE NOT nodeEmpty(el) DO
    genExpr(nodeFirst(el));
    el := nodeRest(el);
  END;
END genExprList;

PROCEDURE genExpr(n:node);
BEGIN
  CASE nodeClass(n) OF
    nUnop: genExpr(nodeArg(n));
           writeOp(nodeOp(n));
    | nOp:  IF (nodeOp(n) = And) OR (nodeOp(n) = Or) THEN
             genLogicalOp(n);
           ELSE
             genExpr(nodeLeftArg(n));
             genExpr(nodeRightArg(n));
             writeOp(nodeOp(n));
           END;
    | nInt: writeInt(nodeInt(n));
    | nBool: writeBool(nodeBool(n));
    | nChar: writeChar(nodeChar(n));
    | nSymbol: writeSymbol(nodeSymbol(n));
    | nCall: genCallStmt(n);
  ELSE
    MyTerminal.fatal("genExpr: unknown expression type");
  END;
END genExpr;

PROCEDURE genLogicalOp(n:node);
(* AND's and OR's end up here. We generate code to evaluate only the first
   if possible. *)
VAR label1, label2:label;
BEGIN
  label1 := newLabel();
  label2 := newLabel();
  genExpr(nodeLeftArg(n));
  IF nodeOp(n) = And THEN (* we branch to FALSE if the first was FALSE *)
    writeCondBranch(Equal, label1);
  ELSE (* it's OR; we branch to TRUE if the first was TRUE *)
    writeCondBranch(Greater, label1);
  END;
  genExpr(nodeRightArg(n)); (* if the first one failed to decide, the value
                             of the 2nd is the value of the whole thing.
  *)
  writeBranch(label2);
  writeLabel(label1);
  writeBool(nodeOp(n) = Or); (* write TRUE if OR, FALSE if AND *)

```



```

        writeLabel(label2);
END genLogicalOp;

BEGIN
END CodeGen.

+++++++
Start CodeWrite.DEF
+++++++
DEFINITION MODULE CodeWrite;

(* This module outputs the code for the SIMPL compiler. *)

FROM Symbol IMPORT symbol;
FROM Token IMPORT tokenClass;

EXPORT QUALIFIED writeLabel, writeStringLabel, writeRoutineLabel, writeHalt,
    writeStringBranch, writeBranch, writeCondBranch, writePop, writeCall,
    writeWriteInt, writeReadInt, writeReturn, writeFReturn, writeOp,
    writeInt, writeBool, writeSymbol, writeChar,
    writeWriteChar, writeReadChar;

PROCEDURE writeLabel(c:CARDINAL);
(* Writes an "L" followed by the number, then a colon. *)

PROCEDURE writeStringLabel(s:ARRAY OF CHAR);
(* Just writes the string followed by a colon. *)

PROCEDURE writeRoutineLabel(routine:symbol);
(* Writes the name of the routine followed by a colon, and writes (on the
    screen) the procedure name so the user knows it's being compiled. *)

PROCEDURE writeStringBranch(s: ARRAY OF CHAR);
(* Write a branch followed by the string *)

PROCEDURE writeCondBranch(tc:tokenClass; c:CARDINAL);
(* Write a conditional branch (Equal, Greater or Less) followed by "L", then
    the number. *)

PROCEDURE writeBranch(c:CARDINAL);
(* Write an unconditional branch to the label. *)

PROCEDURE writePop(s:symbol);
(* Generate the appropriate pop instruction for the symbol *)

PROCEDURE writeCall(s:symbol);
(* Generate a call instruction with the symbol *)

PROCEDURE writeWriteInt;
PROCEDURE writeReadInt;

PROCEDURE writeWriteChar;
PROCEDURE writeReadChar;
(* instructions for I/O *)

PROCEDURE writeReturn(numFormals:CARDINAL);
PROCEDURE writeFReturn(numFormals:CARDINAL);
(* Two types of return instructions; both take the number of formals as arg.
    *)

PROCEDURE writeOp(tc:tokenClass);
(* Write the instruction corresponding to the operator *)

PROCEDURE writeInt(i:INTEGER);
PROCEDURE writeBool(b:BOOLEAN);
PROCEDURE writeChar(c:CHAR);
(* Write pushes for these constants. *)

PROCEDURE writeSymbol(s:symbol);
(* Generate the appropriate push instruction for the symbol *)

PROCEDURE writeHalt;

```

(continued)

January

END CodeWrite.

++++++

Start CodeWrite.MOD

++++++

IMPLEMENTATION MODULE CodeWrite;

```
FROM InOut IMPORT WriteString, WriteLn, WriteInt, WriteCard;
    (* We can't use Write because of a conflict inside tokenClass *)
FROM Symbol IMPORT symbol, symbolString, symbolLexLevel, symbolOffset;
FROM SymbolTable IMPORT currentLexLevel;
FROM Token IMPORT tokenClass, stringType;
IMPORT MyTerminal;
```

```
PROCEDURE writeStringLabel(s:ARRAY OF CHAR);
BEGIN
    WriteString(s);
    WriteString(': ');
END writeStringLabel;
```

```
PROCEDURE writeRoutineLabel(routine:symbol);
VAR name:stringType;
BEGIN
    writeRoutineName(routine);
    WriteString(': ');
    WriteLn;
    symbolString(routine, name);
    MyTerminal.WriteString(name);
    MyTerminal.WriteLineString("...");
END writeRoutineLabel;
```

```
PROCEDURE writeRoutineName(routine:symbol);
VAR name:stringType;
BEGIN
    symbolString(routine, name);
    WriteString(name);
    IF symbolLexLevel(routine) <> 0 THEN
        WriteInt(symbolOffset(routine), 0);
    END;
END writeRoutineName;
```

```
PROCEDURE writeLabel(c:CARDINAL);
BEGIN
    WriteChar('L');
    WriteCard(c, 0);
    WriteChar(':');
    WriteLn;
END writeLabel;
```

```
PROCEDURE writeStringBranch(s:ARRAY OF CHAR);
BEGIN
    writeOpCode('BRANCH ');
    WriteLnString(s);
END writeStringBranch;
```

```
PROCEDURE writeBranch(c:CARDINAL);
BEGIN
    writeOpCode('BRANCH L');
    WriteCard(c, 0);
    WriteLn;
END writeBranch;
```

```
PROCEDURE writeCondBranch(tc:tokenClass; c:CARDINAL);
BEGIN
    CASE tc OF
        Equal: writeOpCode('BREQL L');
        Less: writeOpCode('BRLSS L');
        Greater: writeOpCode('BRGTR L');
    ELSE
        MyTerminal.fatal('writeCondBranch: unknown branch type');
    END;
    WriteCard(c, 0);
    WriteLn;
END writeCondBranch;
```

```
PROCEDURE writeWriteInt;
```



```

BEGIN
    writeOpCode('WRINT');
    WriteLn;
END writeWriteInt;

PROCEDURE writeReadInt;
BEGIN
    writeOpCode('RDINT');
    WriteLn;
END writeReadInt;

PROCEDURE writeWriteChar;
BEGIN
    writeOpCode('WRCHAR');
    WriteLn;
END writeWriteChar;

PROCEDURE writeReadChar;
BEGIN
    writeOpCode('RDCHAR');
    WriteLn;
END writeReadChar;

PROCEDURE writeHalt;
BEGIN
    writeOpCode('HALT');
    WriteLn;
END writeHalt;

PROCEDURE writeReturn(numFormals:CARDINAL);
BEGIN
    writeOpCode('RETURN ');
    WriteCard(numFormals, 0);
    WriteLn;
END writeReturn;

PROCEDURE writeFReturn(numFormals:CARDINAL);
BEGIN
    writeOpCode('FRETURN ');
    WriteCard(numFormals, 0);
    WriteLn;
END writeFReturn;

PROCEDURE writeInt(i:INTEGER);
BEGIN
    writeOpCode('PUSHC ');
    WriteInt(i, 0);
    WriteLn;
END writeInt;

PROCEDURE writeChar(c:CHAR);
BEGIN
    writeOpCode('PUSHC ');
    WriteChar("");
    WriteChar(c);
    WriteLn;
END writeChar;

PROCEDURE writeBool(b:BOOLEAN);
BEGIN
    IF b THEN
        writeInt(1);
    ELSE
        writeInt(0);
    END;
END writeBool;

PROCEDURE writePop(s:symbol);
BEGIN
    writeSymAddr(s, 'POPC ', 'POPL ');
END writePop;

PROCEDURE writeCall(s:symbol);
BEGIN
    writeOpCode("CALL ");

```

(continued)

```

    writeRoutineName(s);
    WriteString(", ");
    WriteInt(currentLexLevel() - symbolLexLevel(s), 0);
    WriteLn;
END writeCall;

PROCEDURE writeSymbol(s:symbol);
BEGIN
    writeSymAddr(s, "PUSH    ", "PUSHL  ");
END writeSymbol;

PROCEDURE writeSymAddr(s:symbol; global, local:ARRAY OF CHAR);
VAR name:stringType;
BEGIN
    IF symbolLexLevel(s) = 0 THEN      (* global variable *)
        symbolString(s, name);
        writeOpCode(global);
        WriteLnString(name);
    ELSE
        writeOpCode(local);
        WriteInt(currentLexLevel() - symbolLexLevel(s), 0);
        WriteString(', ');
        WriteInt(symbolOffset(s), 0);
        symbolString(s, name);
        writeComment(name);
    END;
END writeSymAddr;

PROCEDURE writeOp(tc:tokenClass);
BEGIN
    CASE tc OF
        Plus:      writeOpCode('ADD');
        Minus:     writeOpCode('SUB');
        UMinus:    writeOpCode('NEG');
        Times:     writeOpCode('MUL');
        Divide:    writeOpCode('DIV');
        Not:       writeOpCode('NOT');
        Equal:     writeOpCode('EQUAL');
        Greater:   writeOpCode('GREATER');
        Less:      writeOpCode('LESS');
        NotEqual:  writeOpCode('NOTEQL');
        LessEqual: writeOpCode('LSSEQL');
        GreaterEqual: writeOpCode('GTREQL');
    ELSE MyTerminal.fatal("writeOp: unknown operator");
    END;
    WriteLn;
END writeOp;

PROCEDURE WriteLnString(s:ARRAY OF CHAR);
BEGIN
    WriteString(s);
    WriteLn;
END WriteLnString;

PROCEDURE WriteChar(c:CHAR);
(* can't use InOut.Write, because the name conflicts with the Write in
   tokenClass. *)
VAR s:ARRAY[0..1] OF CHAR;
BEGIN
    s[0] := c;
    s[1] := 0C;
    WriteString(s);
END WriteChar;

PROCEDURE writeOpCode(s:ARRAY OF CHAR);
BEGIN
    WriteString("    ");
    WriteString(s);
END writeOpCode;

PROCEDURE writeComment(s:ARRAY OF CHAR);
BEGIN
    WriteString("    ; ");
    WriteString(s);
    WriteLn;
END writeComment;

```



```
BEGIN
END CodeWrite.
```

```
+++++++
Start Compiler.MOD
+++++++
MODULE Compiler;
```

```
(* A compiler for the SIMPL programming language.
Copyright 1985 By Jonathan Amsterdam.
See the BYTE article "A SIMPL Compiler" for more information.
```

Module map, roughly in order of low to high level:

CharStuff	Low-level character utilities	\	
StringStuff	Low-level string utilities		used in
previous			
MyTerminal	Low-level terminal I/O utilities		projects
LexAnStuff	Toolkit for building lexical analyzers	/	
Token	Token, tokenList and typeType data types		
Symbol	Symbol, symbolList and related data types		
Node	Node and related data types		
TypeChecker	Procedures to do type-checking		
LexAn	Lexical analyzer for compiler		
SymbolTable	Compiler symbol table		
CodeWrite	Actual output of code		
CodeGen	Code generation		
ExprParser	Parses expressions		
Routines	Parses procedure and function declarations		
Parser	Main parser		

The module Debug, also supplied, is not used by the compiler, but contains routines useful in debugging the compiler.

I would appreciate hearing about any bugs in the code. My BIX address is jba.
--Jonathan Amsterdam

*)

```
FROM InOut IMPORT OpenInput, OpenOutput, CloseInput, CloseOutput;
FROM MyTerminal IMPORT ClearScreen, pause, WriteLnString;
FROM Parser IMPORT program;
```

```
BEGIN
  ClearScreen;
  WriteLnString("SIMPL Compiler V1.0");
  OpenInput('SMP');
  OpenOutput('ASM');
  program;
  CloseInput;
  CloseOutput;
  pause('Done--');
END Compiler.
```

```
+++++++
Start ExprParser.DEF
+++++++
DEFINITION MODULE ExprParser;
```

```
(* The part of the parser that handles expressions.
Syntax:
```

```
<expr> ::= <expr> | <relexpr> | <relexpr> OR <expr> | <relexpr> AND <expr>
<relexpr> ::= <intexpr> | <intexpr> <relation> <intexpr>
<intexpr> ::= <term> | <term> + <intexpr> | <term> - <intexpr>
<term> ::= <factor> | <factor> * <term> | <factor> / <term>
<factor> ::= <id> | <number> | <call> | <char>
           - <factor> | NOT <factor> | ( <expr> )
```

*)

```
FROM Node IMPORT node;
```

(continued)

```
EXPORT QUALIFIED expr;
```

```
PROCEDURE expr():node;
```

```
END ExprParser.
```

```
++++++
```

```
Start ExprParser.MOD
```

```
++++++
```

```
IMPLEMENTATION MODULE ExprParser;
```

```
(* Handles parsing of expressions, which are tricky because we have to
   make the operators left-associative, whereas the normal recursive descent
   grammar would have them be right-associative.
```

```
   The problem is that the trees are build from the right. To make
   them get built from the left, we pass to expr, relexpr, intexpr and term
   the partial tree constructed from the left, and each of these procedures
   hooks that tree on to the one it parses in the appropriate way. *)
```

```
FROM Token IMPORT token, tokenClass, isRelation;
FROM LexAn IMPORT getToken, ungetToken, tokenErrorCheck, compError;
FROM Symbol IMPORT symbol, SymbolClass, symbolClassEqual;
FROM SymbolTable IMPORT findSymbol;
FROM Node IMPORT node, emptyNode, makeOpNode, makeUnopNode, makeIntegerNode,
  makeBooleanNode, makeSymbolNode, makeCallNode, makeStringNode, nodeEmpty,
  makeCharNode;
FROM Parser IMPORT actuals;
```

```
CONST dummy = Period;
```

```
(* <expr> ::= <expr> | <relexpr> | <relexpr> OR <expr> | <relexpr> AND <expr> *)
```

```
PROCEDURE expr():node;
```

```
BEGIN
```

```
  RETURN expr1(emptyNode, dummy);
```

```
END expr;
```

```
PROCEDURE expr1(left:node; op:tokenClass):node;
```

```
VAR n:node;
```

```
  t:token;
```

```
BEGIN
```

```
  n := relexpr();
```

```
  getToken(t);
```

```
  IF (t.class = And) OR (t.class = Or) THEN
```

```
    RETURN expr1(buildTree(op, left, n), t.class);
```

```
  ELSE
```

```
    ungetToken;
```

```
    IF nodeEmpty(left) THEN
```

```
      RETURN n;
```

```
    ELSE
```

```
      RETURN makeOpNode(op, left, n);
```

```
    END;
```

```
  END;
```

```
END expr1;
```

```
(* <relexpr> ::= <intexpr> | <intexpr> <relation> <intexpr> *)
```

```
PROCEDURE relexpr():node;
```

```
(* Here we don't have to worry about associativity since relations aren't
   associative! *)
```

```
VAR n:node;
```

```
  t:token;
```

```
BEGIN
```

```
  n := intexpr(emptyNode, dummy);
```

```
  getToken(t);
```

```
  IF isRelation(t.class) THEN
```

```
    RETURN makeOpNode(t.class, n, intexpr(emptyNode, dummy));
```

```
  ELSE
```

```
    ungetToken;
```

```
    RETURN n;
```

```
  END;
```

```
END relexpr;
```

```
(* <intexpr> ::= <term> | <term> + <intexpr> | <term> - <intexpr> *)
```

```
PROCEDURE intexpr(left:node; op:tokenClass):node;
```

```
VAR n:node;
```

```
  t:token;
```



```

BEGIN
  n := term(emptyNode, dummy);
  getToken(t);
  IF (t.class = Plus) OR (t.class = Minus) THEN
    RETURN intexpr(buildTree(op, left, n), t.class);
  ELSE
    ungetToken;
    IF nodeEmpty(left) THEN
      RETURN n;
    ELSE
      RETURN makeOpNode(op, left, n);
    END;
  END;
END intexpr;
(* <term> ::= <factor> | <factor> * <term> | <factor> / <term> *)
PROCEDURE term(left:node; op:tokenClass):node;
VAR n:node;
    t:token;
BEGIN
  n := factor();
  getToken(t);
  IF (t.class = Times) OR (t.class = Divide) THEN
    RETURN term(buildTree(op, left, n), t.class);
  ELSE
    ungetToken;
    IF nodeEmpty(left) THEN
      RETURN n;
    ELSE
      RETURN makeOpNode(op, left, n);
    END;
  END;
END term;

(* <factor> ::= <id> | <number> | <call> | <char> | - <factor> | NOT <factor>
| ( <expr> ) *)
PROCEDURE factor():node;
VAR n:node;
    t:token;
BEGIN
  getToken(t);
  CASE t.class OF
    Int: RETURN makeIntegerNode(t.integer);
    Character: RETURN makeCharNode(t.ch);
    String: RETURN makeStringNode(t.string);
    Minus: RETURN makeUnopNode(UMinus, factor());
    Not: RETURN makeUnopNode(Not, factor());
    True: RETURN makeBooleanNode(TRUE);
    False: RETURN makeBooleanNode(FALSE);
    Lparen:
      n := expr();
      tokenErrorCheck(Rparen, 'Right paren expected');
      RETURN n;
    Identifier: RETURN callOrId(t);
  ELSE
    compError('bad factor');
    RETURN emptyNode;
  END;
END factor;

PROCEDURE callOrId(t:token):node;
VAR s:symbol;
BEGIN
  s := findSymbol(t.string);
  IF symbolClassEqual(s, Func) THEN
    RETURN makeCallNode(s, actuals());
  ELSIF symbolClassEqual(s, Proc) THEN
    compError('procedures cannot be used in an expression');
    RETURN emptyNode;
  ELSE (* it's a variable *)
    RETURN makeSymbolNode(s);
  END;
END callOrId;

```

(continued)

```

PROCEDURE buildTree(op:tokenClass; n1, n2:node):node;
(* This is the key hack that builds trees up from the left if necessary. *)
BEGIN
  IF nodeEmpty(n1) THEN
    RETURN n2;
  ELSE
    RETURN makeOpNode(op, n1, n2);
  END;
END buildTree;

BEGIN
END ExprParser.

+++++++
Start LexAn.DEF
+++++++
DEFINITION MODULE LexAn;

(* The lexical analyzer for the SIMPL compiler. It uses the InOut module
do input, so you can get input from a file by redirecting it with
InOut.OpenInput.
This module also handles errors. *)

FROM Token IMPORT token, tokenClass;

EXPORT QUALIFIED getToken, ungetToken, getTokenClass, tokenErrorCheck,
  getTokenErrorCheck, errorFlag, compError, peekTokenClass;

VAR errorFlag:BOOLEAN;      (* Set to TRUE when an error occurs. *)

PROCEDURE getToken(VAR t:token);
(* Get a token from the input stream. *)

PROCEDURE ungetToken;
(* Push a token back on the input stream. Can only unget one at a time. *)

PROCEDURE getTokenClass():tokenClass;
(* Get a token from the input stream, but just return its class. *)

PROCEDURE peekTokenClass():tokenClass;
(* Get a token from the input stream, unget it, and return its class. *)

PROCEDURE tokenErrorCheck(tc:tokenClass; msg: ARRAY OF CHAR);
(* Read a token from the input stream and compare its class to tc. If they
are the same, do nothing. If they are different, write the current line
to the screen, print the message and unget the token. *)

PROCEDURE getTokenErrorCheck(VAR t:token; tc:tokenClass; msg: ARRAY OF CHAR);
(* Like tokenErrorCheck, but returns the token as well. *)

PROCEDURE compError(msg:ARRAY OF CHAR);
(* Writes the current line and displays msg. Sets errorFlag to TRUE. *)

END LexAn.

+++++++
Start Lexan.MOD
+++++++

IMPLEMENTATION MODULE LexAn;

(* Lexical analyzer for the SIMPL compiler. Uses the routines in
LexAnStuff. *)

FROM InOut IMPORT EOL;
FROM Token IMPORT token, tokenClass;
FROM MyTerminal IMPORT fatal, WriteLnString;
FROM StringStuff IMPORT stringLen;
FROM SymbolTable IMPORT enterKeyword, findKeyword;
FROM LexAnStuff IMPORT dispatch, enterAll, enterChar, enterEndOfFile, ignore,
  enterAlphas, enterDigits, skipToChar, alphaNumString, posInteger, string,
  enterWhite, writeLine, getChar, ungetChar;

VAR tok: token;
    ungotten: BOOLEAN;

```



```

PROCEDURE getToken(VAR t:token);
BEGIN
    getTok;
    t := tok;
END getToken;

PROCEDURE getTok;
VAR c:CHAR;
BEGIN
    IF ungotten THEN
        ungotten := FALSE;
    ELSE
        dispatch;
    END;
END getTok;

PROCEDURE ungetToken;
BEGIN
    IF ungotten THEN
        fatal("ungetToken: can only unget one token at a time");
    ELSE
        ungotten := TRUE;
    END;
END ungetToken;

PROCEDURE getTokenClass():tokenClass;
BEGIN
    getTok;
    RETURN tok.class;
END getTokenClass;

PROCEDURE peekTokenClass():tokenClass;
BEGIN
    getTok;
    ungetToken;
    RETURN tok.class;
END peekTokenClass;

PROCEDURE tokenErrorCheck(tc:tokenClass; msg: ARRAY OF CHAR);
BEGIN
    getTok;
    IF tok.class <> tc THEN
        compError(msg);
        IF tok.class = EndOfInput THEN
            fatal("unexpected end of input");
        END;
        ungetToken;
    END;
END tokenErrorCheck;

PROCEDURE getTokenErrorCheck(VAR t:token; tc:tokenClass; msg: ARRAY OF CHAR);
BEGIN
    tokenErrorCheck(tc, msg);
    t := tok;
END getTokenErrorCheck;

    (** reading procedures **)

PROCEDURE illegalChar(c:CHAR);
VAR charstring:ARRAY[0..1] OF CHAR;
BEGIN
    charstring[0] := c;          (* fake a 1-char string *)
    charstring[1] := 0C;
    compError('illegal character');
    getTok;
END illegalChar;

PROCEDURE comment(c:CHAR);      (* Comments are ignored. They are delimited
                                by { and } *)
BEGIN
    skipToChar('}');
    getTok;
END comment;

```

(continued)

```

PROCEDURE idOrKeyword(c:CHAR);
(* Get an alphanumeric string from the input. If we find it in the symbol
   table marked as a keyword, then it's a keyword; findKeyword will have taken
   care of setting tok.class to the right value. Else, it's an identifier. *)
BEGIN
  IF NOT alphaNumString(c, tok.string) THEN
    compError('identifier too long');
  END;
  IF NOT findKeyword(tok.string, tok.class) THEN
    tok.class := Identifier;
  END;
END idOrKeyword;

PROCEDURE posInt(c:CHAR);
BEGIN
  tok.class := Int;
  tok.integer := posInteger(c);
END posInt;

PROCEDURE charProc(c:CHAR);
(* Read a character, delimited by delim, from the input. Can use
   backslash: \n = newline, \t = tab, anything else literal. *)
BEGIN
  tok.class := Character;
  IF (NOT string(c, tok.string)) OR (stringLen(tok.string) > 1) THEN
    compError('illegal character constant');
  END;
  tok.ch := tok.string[0];
END charProc;

PROCEDURE stringProc(c:CHAR);
(* Read a string from the input. If too long, skip to the next delim. *)
BEGIN
  tok.class := String;
  IF NOT string(c, tok.string) THEN
    compError('string too long');
    skipToChar(c);
    c := getChar();      (* get the delimiter *)
  END;
END stringProc;

      (***) Reading special characters (***)

PROCEDURE period(c:CHAR); BEGIN tok.class := Period; END period;
PROCEDURE semicolon(c:CHAR); BEGIN tok.class := Semicolon; END semicolon;
PROCEDURE equal(c:CHAR); BEGIN tok.class := Equal; END equal;
PROCEDURE comma(c:CHAR); BEGIN tok.class := Comma; END comma;
PROCEDURE plus(c:CHAR); BEGIN tok.class := Plus; END plus;
PROCEDURE minus(c:CHAR); BEGIN tok.class := Minus; END minus;
PROCEDURE times(c:CHAR); BEGIN tok.class := Times; END times;
PROCEDURE divide(c:CHAR); BEGIN tok.class := Divide; END divide;
PROCEDURE lparen(c:CHAR); BEGIN tok.class := Lparen; END lparen;
PROCEDURE rparen(c:CHAR); BEGIN tok.class := Rparen; END rparen;

PROCEDURE greater(c:CHAR);
BEGIN
  IF getChar() = '=' THEN
    tok.class := GreaterEqual;
  ELSE
    ungetChar;
    tok.class := Greater;
  END;
END greater;

PROCEDURE less(c:CHAR);
BEGIN
  c := getChar();
  IF c = '=' THEN
    tok.class := LessEqual;
  ELSIF c = '>' THEN
    tok.class := NotEqual;
  ELSE
    ungetChar;
    tok.class := Less;
  END;
END less;

```



```

PROCEDURE colon(c:CHAR);
BEGIN
  IF getChar() = '=' THEN
    tok.class := Assignment;
  ELSE
    ungetChar;
    tok.class := Colon;
  END;
END colon;

```

```

PROCEDURE endOfInput(c:CHAR);
BEGIN
  tok.class := EndOfInput;
END endOfInput;

```

(** initialization of charTable **)

```

PROCEDURE initCharTable;
BEGIN
  enterAll(illegalChar);
  enterWhite(ignore);
  enterAlphas(idOrKeyword);
  enterDigits(posInt);
  enterChar('.', period);
  enterChar(':', colon);
  enterChar(';', semicolon);
  enterChar('(', lparen);
  enterChar(')', rparen);
  enterChar(',', comma);
  enterChar('=', equal);
  enterChar('>', greater);
  enterChar('<', less);
  enterChar('+', plus);
  enterChar('-', minus);
  enterChar('*', times);
  enterChar('/', divide);
  enterChar('{', comment);
  enterChar('"', stringProc);
  enterChar("'", charProc);
  enterEndOfFile(endOfInput);
END initCharTable;

```

```

PROCEDURE enterKeywords;
BEGIN
  enterKeyword('AND', And);
  enterKeyword('BEGIN', Begin);
  enterKeyword('BOOLEAN', Boolean);
  enterKeyword('CHAR', Char);
  enterKeyword('DO', Do);
  enterKeyword('ELSE', Else);
  enterKeyword('ELSIF', Elif);
  enterKeyword('END', End);
  enterKeyword('FALSE', False);
  enterKeyword('FUNCTION', Function);
  enterKeyword('IF', If);
  enterKeyword('INTEGER', Integer);
  enterKeyword('NOT', Not);
  enterKeyword('OR', Or);
  enterKeyword('PROCEDURE', Procedure);
  enterKeyword('PROGRAM', Program);
  enterKeyword('READ', Read);
  enterKeyword('RETURN', Return);
  enterKeyword('THEN', Then);
  enterKeyword('TRUE', True);
  enterKeyword('VAR', Var);
  enterKeyword('WHILE', While);
  enterKeyword('WRITE', Write);
END enterKeywords;

```

(** errors **)

```

PROCEDURE compError(msg:ARRAY OF CHAR);
BEGIN
  writeLine;
  WriteLnString(msg);

```

(continued)

January

```
errorFlag := TRUE;
END compError;
```

```
BEGIN
  ungotten := FALSE;
  errorFlag := FALSE;
  initCharTable;
  enterKeywords;
END LexAn.
```

```
+++++++
Start Node.DEF
+++++++
DEFINITION MODULE Node;
```

```
(* Nodes are what make up the parse tree produced by the SIMPL parser.
   They consist of all data relevant to generating code. *)
```

```
FROM Token IMPORT tokenClass, typeType;
FROM Symbol IMPORT symbol;
```

```
EXPORT QUALIFIED node, nodeClass, NodeClass, emptyNode, nodeEmpty, freeNode,
  makeStmtsNode, makeIfNode, makeWhileNode, makeReturnNode,
  makeAssignmentNode, makeExprListNode, makeOpNode, makeUnopNode,
  makeIntegerNode, makeBooleanNode, makeSymbolNode, makeCallNode,
  makeWriteNode, makeReadNode, makeStringNode, makeCharNode,
  nodeFirst, nodeRest, nodeTest, nodeThen, nodeElse, nodeStmts, nodeRHS,
  nodeLHS, nodeArgs, nodeRoutine, nodeExpr, nodeArg, nodeLeftArg,
  nodeRightArg, nodeOp, nodeSymbol, nodeType, nodeInt, nodeBool,
  nodeNumFormals, nodeString, nodeChar;
```

```
TYPE
  NodeClass = (* the different kinds of nodes *)
    (nOp, (* binary operators (+, -, *, /, relations, AND, OR)
*)
    nUnop, (* unary operators (unary minus, NOT) *)
    nBool, (* a boolean constant (TRUE, FALSE) *)
    nInt, (* an integer constant *)
    nChar, (* a character constant *)
    nString, (* a string constant *)
    nSymbol, (* a symbol (variable) *)
    nIf, (* IF statement *)
    nWhile, (* WHILE statement *)
    nReturn, (* RETURN statement, either procedure or function *)
    nCall, (* procedure call (statement) or function call *)
    nAssignment, (* assignment statement *)
    nWrite, nRead, (* WRITE and READ statements *)
    nList); (* a list of statements or expressions *)
```

```
node;
```

```
VAR emptyNode: node;
```

```
PROCEDURE nodeClass(n:node):NodeClass;
(* Returns the class of node *)
```

```
PROCEDURE nodeEmpty(n:node):BOOLEAN;
(* Returns true if node is the emptyNode *)
```

```
PROCEDURE freeNode(n:node);
(* Frees the storage associated with n *)
```

```
(*** Node creation ***)
```

```
PROCEDURE makeStmtsNode(first, rest:node):node;
(* Make a node representing a list of statements *)
```

```
PROCEDURE makeReturnNode(routine:symbol; returnExpr:node):node;
(* Make a return node. Routine is the routine we are returning from.
   returnExpr is an expression to be returned, for functions; for procedures,
   it should be the empty node. *)
```

```
PROCEDURE makeCallNode(name:symbol; actuals:node):node;
PROCEDURE makeWriteNode(actuals:node):node;
PROCEDURE makeReadNode(actuals:node):node;
(* In all of these, actuals should have been made with makeExprListNode. *)
```



```

PROCEDURE makeIfNode(test, then, else:node):node;
PROCEDURE makeWhileNode(test, stmts:node):node;
PROCEDURE makeAssignmentNode(var:symbol; expr:node):node;
PROCEDURE makeExprListNode(first, rest:node):node;
PROCEDURE makeOpNode(op:tokenClass; leftarg, rightarg:node):node;
PROCEDURE makeUnopNode(op:tokenClass; arg:node):node;
PROCEDURE makeIntegerNode(i:INTEGER):node;
PROCEDURE makeBooleanNode(b:BOOLEAN):node;
PROCEDURE makeSymbolNode(id:symbol):node;
PROCEDURE makeStringNode(s:ARRAY OF CHAR):node;
PROCEDURE makeCharNode(c:CHAR):node;

    (** Accessing parts of nodes **)

(* many nodes have a type associated with them *)
PROCEDURE nodeType(n:node):typeType;

(* for constants *)
PROCEDURE nodeInt(n:node):INTEGER;
PROCEDURE nodeBool(n:node):BOOLEAN;
PROCEDURE nodeChar(n:node):CHAR;
PROCEDURE nodeString(n:node; VAR s:ARRAY OF CHAR);
(* Just truncates if s is too short. *)

(* for lists *)
PROCEDURE nodeFirst(n:node):node;
PROCEDURE nodeRest(n:node):node;

(* for IF statements *)
PROCEDURE nodeTest(n:node):node;          (* also for WHILE statements *)
PROCEDURE nodeThen(n:node):node;
PROCEDURE nodeElse(n:node):node;

(* for WHILE statements *)
PROCEDURE nodeStmts(n:node):node;

(* for assignment statements *)
PROCEDURE nodeRHS(n:node):node;          (* right-hand side *)
PROCEDURE nodeLHS(n:node):symbol;        (* left-hand side *)

(* for calls *)
PROCEDURE nodeArgs(n:node):node;
PROCEDURE nodeRoutine(n:node):symbol;

(* for RETURN statements *)
PROCEDURE nodeExpr(n:node):node;
PROCEDURE nodeNumFormals(n:node):CARDINAL;

(* for ops and unops *)
PROCEDURE nodeArg(n:node):node;
PROCEDURE nodeLeftArg(n:node):node;
PROCEDURE nodeRightArg(n:node):node;
PROCEDURE nodeOp(n:node):tokenClass;

(* for symbols *)
PROCEDURE nodeSymbol(n:node):symbol;

END Node.

++++++
Start Node.MOD
++++++

IMPLEMENTATION MODULE Node;

(* Procedures for constructing and manipulating the nodes of the parse tree.
   Most type-checking is done here. *)
FROM Token IMPORT tokenClass, typeType, stringType, isRelation;
FROM Symbol IMPORT symbol, SymbolClass, symbolType, emptySymbol, numFormals;
FROM Storage IMPORT ALLOCATE, DEALLOCATE;
FROM TypeChecker IMPORT typeCompatible, opAppropriate, callCheck, unopCheck,
    readCheck, writeCheck, binopCheck, returnCheck, assignCheck;
FROM MyTerminal IMPORT WriteString, fatal;
FROM StringStuff IMPORT stringCopy;

```

(continued)

TYPE

```

node = POINTER TO nodeRec;
nodeRec = RECORD
    type: typeType;
    CASE class: NodeClass OF
        nOp: op: tokenClass; leftArg, rightArg: node;
        nUnop: unop: tokenClass; arg: node;
        nBool: bool: BOOLEAN;
        nInt: int: INTEGER;
        nChar: ch: CHAR;
        nString: string: stringType;
        nSymbol: sym: symbol;
        nIf: test, then, else: node;
        nWhile: wtest, stmts: node;
        nAssignment: LHS: symbol; RHS: node;
        nCall, nWrite, nRead: routine: symbol; args: node;
        nReturn: nFormals: CARDINAL; expr: node;
        nList: first, rest: node;
    END;
END;

```

```

PROCEDURE nodeClass(n: node): NodeClass;

```

```

BEGIN
    RETURN n^.class;
END nodeClass;

```

```

PROCEDURE nodeEmpty(n: node): BOOLEAN;

```

```

BEGIN
    RETURN n = emptyNode;
END nodeEmpty;

```

```

PROCEDURE freeNode(n: node);

```

```

BEGIN
    IF n <> emptyNode THEN
        WITH n^ DO CASE class OF
            nInt, nBool, nSymbol, nString, nChar: (* do nothing *);
            | nOp: freeNode(leftArg);
                freeNode(rightArg);
            | nUnop: freeNode(arg);
            | nIf: freeNode(test);
                freeNode(then);
                freeNode(else);
            | nWhile: freeNode(wtest);
                freeNode(stmts);
            | nAssignment: freeNode(RHS);
            | nCall, nRead, nWrite: freeNode(args);
            | nReturn: freeNode(expr);
            | nList: freeNode(first);
                freeNode(rest);
            ELSE
                WriteString("freeNode: unknown node type");
            END; END;
            DISPOSE(n); (* , n^.class); *)
        END;
    END freeNode;

```

```

(***) node creation (***)

```

```

PROCEDURE makeStmtsNode(first, rest: node): node;

```

```

VAR n: node;
BEGIN
    n := newNode(nList);
    n^.first := first;
    n^.rest := rest;
    RETURN n;
END makeStmtsNode;

```

```

PROCEDURE makeExprListNode(first, rest: node): node;

```

```

VAR n: node;
BEGIN
    n := newNode(nList);
    n^.first := first;
    n^.rest := rest;
    RETURN n;
END makeExprListNode;

```



```

PROCEDURE makeIfNode(test, then, else:node):node;
VAR n:node;
BEGIN
    n := newNode(nIf);
    n^.test := test;
    n^.then := then;
    n^.else := else;
    RETURN n;
END makeIfNode;

PROCEDURE makeWhileNode(test, stmts:node):node;
VAR n:node;
BEGIN
    n := newNode(nWhile);
    n^.wtest := test;
    n^.stmts := stmts;
    RETURN n;
END makeWhileNode;

PROCEDURE makeReturnNode(routine:symbol; returnExpr:node):node;
VAR n:node;
BEGIN
    n := newNode(nReturn);
    n^.expr := returnExpr;
    IF returnCheck(routine, returnExpr) THEN
        n^.nFormals := numFormals(routine);
    END;
    RETURN n;
END makeReturnNode;

PROCEDURE makeAssignmentNode(var:symbol; expr:node):node;
VAR n:node;
BEGIN
    n := newNode(nAssignment);
    n^.LHS := var;
    n^.RHS := expr;
    assignCheck(var, expr);
    RETURN n;
END makeAssignmentNode;

PROCEDURE makeOpNode(op:tokenClass; leftarg, rightarg:node):node;
VAR n:node;
    typeOK:BOOLEAN;
BEGIN
    n := newNode(nOp);
    n^.op := op;
    n^.leftArg := leftarg;
    n^.rightArg := rightarg;
    typeOK := binopCheck(op, leftarg, rightarg);
    IF isRelation(op) THEN
        n^.type := tBoolean;
    ELSIF typeOK THEN
        n^.type := leftarg^.type;
    ELSE
        n^.type := tUnknown;
    END;
    RETURN n;
END makeOpNode;

PROCEDURE makeUnopNode(op:tokenClass; arg:node):node;
VAR n:node;
BEGIN
    n := newNode(nUnop);
    n^.unop := op;
    n^.arg := arg;
    IF unopCheck(op, arg) THEN
        n^.type := arg^.type;
    ELSE
        n^.type := tUnknown;
    END;
    RETURN n;
END makeUnopNode;

PROCEDURE makeIntegerNode(i:INTEGER):node;
VAR n:node;

```

(continued)

```

BEGIN
    n := newNode(nInt);
    n^.type := tInteger;
    n^.int := i;
    RETURN n;
END makeIntegerNode;

PROCEDURE makeBooleanNode(b:BOOLEAN):node;
VAR n:node;
BEGIN
    n := newNode(nBool);
    n^.type := tBoolean;
    n^.bool := b;
    RETURN n;
END makeBooleanNode;

PROCEDURE makeCharNode(c:CHAR):node;
VAR n:node;
BEGIN
    n := newNode(nChar);
    n^.type := tChar;
    n^.ch := c;
    RETURN n;
END makeCharNode;

PROCEDURE makeSymbolNode(s:symbol):node;
VAR n:node;
BEGIN
    n := newNode(nSymbol);
    n^.type := symbolType(s);
    n^.sym := s;
    RETURN n;
END makeSymbolNode;

PROCEDURE makeCallNode(name:symbol; actuals:node):node;
VAR n:node;
BEGIN
    n := newNode(nCall);
    WITH n^ DO
        routine := name;
        args := actuals;
        type := symbolType(name);
        callCheck(routine, args);
    END;
    RETURN n;
END makeCallNode;

PROCEDURE makeWriteNode(actuals:node):node;
VAR n:node;
BEGIN
    writeCheck(actuals);
    n := newNode(nWrite);
    n^.routine := emptySymbol;
    n^.args := actuals;
    RETURN n;
END makeWriteNode;

PROCEDURE makeReadNode(actuals:node):node;
VAR n:node;
BEGIN
    readCheck(actuals);
    n := newNode(nRead);
    n^.routine := emptySymbol;
    n^.args := actuals;
    RETURN n;
END makeReadNode;

PROCEDURE makeStringNode(s:ARRAY OF CHAR):node;
VAR n:node;
BEGIN
    n := newNode(nString);
    stringCopy(n^.string, s);
    RETURN n;
END makeStringNode;

PROCEDURE newNode(nc:NodeClass):node;

```



```

VAR n:node;
BEGIN
    NEW(n); (* nc); *)
    n^.class := nc;
    n^.type := tUnknown;
    RETURN n;
END newNode;

                (** node access **)

PROCEDURE nodeInt(n:node):INTEGER;
BEGIN
    nodeClassCheck('nodeInt', n, nInt);
    RETURN n^.int;
END nodeInt;

PROCEDURE nodeBool(n:node):BOOLEAN;
BEGIN
    nodeClassCheck('nodeBool', n, nBool);
    RETURN n^.bool;
END nodeBool;

PROCEDURE nodeChar(n:node):CHAR;
BEGIN
    nodeClassCheck('nodeChar', n, nChar);
    RETURN n^.ch;
END nodeChar;

PROCEDURE nodeString(n:node; VAR s:ARRAY OF CHAR);
BEGIN
    nodeClassCheck('nodeString', n, nString);
    stringCopy(s, n^.string);
END nodeString;

PROCEDURE nodeFirst(n:node):node;
BEGIN
    nodeClassCheck('nodeFirst', n, nList);
    RETURN n^.first;
END nodeFirst;

PROCEDURE nodeRest(n:node):node;
BEGIN
    nodeClassCheck('nodeRest', n, nList);
    RETURN n^.rest;
END nodeRest;

PROCEDURE nodeTest(n:node):node;
BEGIN
    IF n^.class = nIf THEN
        RETURN n^.test;
    ELSIF n^.class = nWhile THEN
        RETURN n^.wtest;
    ELSE
        nodeClassError('nodeTest');
        RETURN emptyNode;
    END;
END nodeTest;

PROCEDURE nodeThen(n:node):node;
BEGIN
    nodeClassCheck('nodeThen', n, nIf);
    RETURN n^.then;
END nodeThen;

PROCEDURE nodeElse(n:node):node;
BEGIN
    nodeClassCheck('nodeElse', n, nIf);
    RETURN n^.else;
END nodeElse;

PROCEDURE nodeStmts(n:node):node;
BEGIN
    nodeClassCheck('nodeStmts', n, nWhile);
    RETURN n^.stmts;
END nodeStmts;

```

(continued)

January

```
PROCEDURE nodeRHS(n:node):node;
BEGIN
    nodeClassCheck('nodeRHS', n, nAssignment);
    RETURN n^.RHS;
END nodeRHS;

PROCEDURE nodeLHS(n:node):symbol;
BEGIN
    nodeClassCheck('nodeLHS', n, nAssignment);
    RETURN n^.LHS;
END nodeLHS;

PROCEDURE nodeArgs(n:node):node;
BEGIN
    WITH n^ DO
        IF (class = nCall) OR (class = nRead) OR (class = nWrite) THEN
            RETURN args;
        ELSE
            nodeClassError('nodeArgs');
        END;
    END;
END nodeArgs;

PROCEDURE nodeRoutine(n:node):symbol;
BEGIN
    nodeClassCheck('nodeRoutine', n, nCall);
    RETURN n^.routine;
END nodeRoutine;

PROCEDURE nodeExpr(n:node):node;
BEGIN
    nodeClassCheck('nodeExpr', n, nReturn);
    RETURN n^.expr;
END nodeExpr;

PROCEDURE nodeArg(n:node):node;
BEGIN
    nodeClassCheck('nodeArg', n, nUnop);
    RETURN n^.arg;
END nodeArg;

PROCEDURE nodeLeftArg(n:node):node;
BEGIN
    nodeClassCheck('nodeLeftArg', n, nOp);
    RETURN n^.leftArg;
END nodeLeftArg;

PROCEDURE nodeRightArg(n:node):node;
BEGIN
    nodeClassCheck('nodeRightArg', n, nOp);
    RETURN n^.rightArg;
END nodeRightArg;

PROCEDURE nodeOp(n:node):tokenClass;
BEGIN
    IF n^.class = nOp THEN
        RETURN n^.op;
    ELSIF n^.class = nUnop THEN
        RETURN n^.unop;
    ELSE
        nodeClassError('nodeOp');
        RETURN Plus;
    END;
END nodeOp;

PROCEDURE nodeSymbol(n:node):symbol;
BEGIN
    nodeClassCheck('nodeSymbol', n, nSymbol);
    RETURN n^.sym;
END nodeSymbol;

PROCEDURE nodeNumFormals(n:node):CARDINAL;
BEGIN
    nodeClassCheck('nodeNumFormals', n, nReturn);
    RETURN n^.nFormals;
END nodeNumFormals;
```



```

PROCEDURE nodeType(n:node):typeType;
BEGIN
    RETURN n^.type;
END nodeType;

(** other **)

PROCEDURE nodeClassCheck(s:ARRAY OF CHAR; n:node; nc:NodeClass);
BEGIN
    IF n^.class <> nc THEN
        nodeClassError(s);
    END;
END nodeClassCheck;

PROCEDURE nodeClassError(s:ARRAY OF CHAR);
BEGIN
    WriteString(s);
    fatal(": node of wrong type");
END nodeClassError;

BEGIN
    emptyNode := NIL;
END Node.

+++++++
Start Parser.DEF
+++++++
DEFINITION MODULE Parser;

(* This is the bulk of the SIMPL parser. It covers most of the language.
   For routines (procedures and functions) see Routines.
   For expressions, see ExprParser.

   Syntax handled by this module:

<program> ::= PROGRAM <id>; <vars> <routines> <block> .

<vars> ::= <empty> | VAR <varlist>
<varlist> ::= <decl> | <decl> <varlist>
<decl> ::= <idlist> : <type> ;
<idlist> ::= <id> | <id> , <idlist>
<type> ::= INTEGER | BOOLEAN | CHAR

<block> ::= BEGIN <stmts> END
<stmts> ::= <empty> | <stmt> ; <stmts>
<stmt> ::= <while> | <if> | <return> | <assign> | <call>
<while> ::= WHILE <expr> DO <stmts> END
<if> ::= IF <elsif> END
<elsif> ::= <expr> THEN <stmts> <else>
<else> ::= <empty> | ELSIF <elsif> | ELSE <stmts>
<return> ::= RETURN | RETURN <expr>
<assign> ::= <id> := <expr>
<call> ::= <id> <actuals>
<actuals> ::= <empty> | ( <exprlist> )
<exprlist> ::= <expr> | <expr> , <exprlist>
*)

FROM Symbol IMPORT symbol;
FROM Node IMPORT node;
FROM Token IMPORT tokenList;

EXPORT QUALIFIED program, vars, idlist, block, actuals;

PROCEDURE program;
(* Parse the entire program *)

PROCEDURE vars(routineName:symbol);
(* Parse variable declarations. Scope indicates whether these are local
   or global variables. RoutineName is the name of the routine currently
   being compiled; if these are global variables, it should be the name of
   the program. *)

PROCEDURE idlist():tokenList;
(* Parse a list of identifiers *)

```

(continued)

```

PROCEDURE block(routine:symbol):node;
(* Parse a block of code. Routine is the routine currently being compiled. *)

PROCEDURE actuals():node;
(* Parse a list of actual parameters, i.e. a list of expressions. *)

END Parser.

+++++++
Start Parser.MOD
+++++++
IMPLEMENTATION MODULE Parser;

(* Most of the parser for the SIMPL compiler. It is a top-down, recursive
descent parser. *)

FROM Token IMPORT token, tokenClass, isType, emptyTokenList,
  t1Token, t1Next, t1Empty, addToTokenList, tokenList, freeTokenList,
  typeType, tokenClassToType;
FROM LexAn IMPORT getToken, getTokenClass, peekTokenClass, compError,
  ungetToken, tokenErrorCheck, getTokenErrorCheck;
FROM Symbol IMPORT symbol, emptySymbol, SymbolClass, symbolEmpty,
  symbolClassEqual, symbolEqual;
FROM SymbolTable IMPORT enterSymbol, enterLocal, enterFormal, findSymbol,
  currentLexLevel;
FROM Node IMPORT node, emptyNode, makeStmtsNode, makeIfNode, makeWhileNode,
  makeReturnNode, makeAssignmentNode, makeExprListNode,
  makeCallNode, makeReadNode, makeWriteNode, nodeType;
FROM CodeGen IMPORT genBlock, genGlobal;
FROM CodeWrite IMPORT writeStringBranch, writeHalt, writeRoutineLabel;
FROM TypeChecker IMPORT boolCheck;
FROM ExprParser IMPORT expr;
FROM Routines IMPORT routines;
FROM MyTerminal IMPORT fatal;

VAR programName:symbol;

(* <program> ::= PROGRAM <id>; <vars> <routines> <block> . *)
PROCEDURE program;
VAR t:token;
  n:node;
BEGIN
  tokenErrorCheck(Program, 'keyword "PROGRAM" expected');
  getTokenErrorCheck(t, Identifier, 'name of program expected');
  IF t.class <> Identifier THEN
    t.string := "???"; (* if the program name isn't given, make one up *)
  END;
  programName := enterSymbol(t.string, Proc, tUnknown);
  writeStringBranch(t.string);
  tokenErrorCheck(Semicolon, 'semicolon expected');
  vars(emptySymbol);
  routines;
  writeRoutineLabel(programName);
  genBlock(block(programName));
  tokenErrorCheck(Period, 'period expected');
  tokenErrorCheck(EndOfInput, 'end of input expected');
  writeHalt;
END program;

(***) variable declarations (***)

(* <vars> ::= <empty> | VAR <varlist> *)
PROCEDURE vars(routineName:symbol);
BEGIN
  IF getTokenClass() = Var THEN
    varlist(routineName);
  ELSE
    ungetToken;
  END;
END vars;

(* <varlist> ::= <decl> | <decl> <varlist>
We can recognize the end of a varlist by seeing if the next token is an
identifier. An Id indicates the varlist continues. If it didn't we'd
see a keyword: either Begin, Procedure or Function. *)

```



```

PROCEDURE varlist(routineName:symbol);
BEGIN
  decl(routineName);
  IF peekTokenClass() = Identifier THEN
    varlist(routineName);
  END;
END varlist;

(* <decl> ::= <idlist> : <type> ;
   Declarations. All the work of putting information about the variables into
   the symbol table is done here. *)
PROCEDURE decl(routineName:symbol);
VAR tl, tokenp:tokenList;
    t, id:token;
    tt:typeType;
BEGIN
  tl := idlist();
  tokenErrorCheck(Colon, 'colon expected');
  getToken(t);
  IF NOT isType(t.class) THEN
    compError('type name expected');
    tt := tUnknown;
    ungetToken;
  ELSE
    tt := tokenClassToType(t.class);
  END;
  tokenErrorCheck(Semicolon, 'semicolon expected');
  tokenp := tl;
  (* Enter the variables into the symbol table. For globals, also generate
   the variables. *)
  WHILE NOT tlEmpty(tokenp) DO
    tlToken(tokenp, id);
    IF currentLexLevel() = 0 THEN
      genGlobal(enterSymbol(id.string, Variable, tt));
    ELSE
      enterLocal(id.string, tt, routineName);
    END;
    tokenp := tlNext(tokenp);
  END;
  freeTokenList(tl);
END decl;

(* <idlist> ::= <id> | <id> , <idlist> *)
PROCEDURE idlist():tokenList;
VAR t: token;
BEGIN
  getTokenErrorCheck(t, Identifier, 'identifier expected');
  IF getTokenClass() <> Comma THEN (* this is the end of the idlist *)
    ungetToken;
    IF t.class = Identifier THEN
      RETURN addToTokenList(t, emptyTokenList);
    ELSE
      RETURN emptyTokenList;
    END;
  (* we saw a comma, so there's more *)
  ELSIF t.class = Identifier THEN
    RETURN addToTokenList(t, idlist());
  ELSE
    RETURN idlist();
  END;
END idlist;

(***) blocks and statements (***)

(* <block> ::= BEGIN <stmts> END *)
PROCEDURE block(routine:symbol):node;
VAR n:node;
BEGIN
  tokenErrorCheck(Begin, 'BEGIN expected');
  n := stmts(routine);
  tokenErrorCheck(End, '"END" expected');
  RETURN n;
END block;

```

(continued)

```

(* <stmts> ::= <empty> | <stmt> ; <stmts>
   We can recognize an empty <stmts> by seeing if the next token is ELSE,
   ELSIF or END. *)
PROCEDURE stmts(routine:symbol):node;
VAR n:node;
    tc:tokenClass;
BEGIN
    tc := peekTokenClass();
    IF (tc = Else) OR (tc = Elsf) OR (tc = End) THEN
        RETURN emptyNode;
    ELSE
        n := stmt(routine);
        tokenErrorCheck(Semicolon, 'a semicolon must end a statement');
        RETURN makeStmtsNode(n, stmts(routine));
    END;
END stmts;

(* <stmt> ::= <while> | <if> | <return> | <assign> | <call> |
   <write> | <read> *)
PROCEDURE stmt(routine:symbol):node;
VAR t:token;
BEGIN
    getToken(t);
    CASE t.class OF
        If: RETURN ifStmt(routine);
        While: RETURN whileStmt(routine);
        Return: RETURN returnStmt(routine);
            IF symbolEqual(routine, programName) THEN
                compError("can't return from main program");
            ELSE
                RETURN makeReturnNode(routine, expr());
            END;
        Write: RETURN makeWriteNode(actuals());
        Read: RETURN makeReadNode(actuals());
        Identifier: RETURN assignOrCallStmt(t);
    ELSE
        compError('illegal statement type');
        RETURN emptyNode;
    END;
END stmt;

(* <if> ::= IF <elsif> END *)
PROCEDURE ifStmt(routine:symbol):node;
VAR n:node;
BEGIN
    n := elsif(routine);
    tokenErrorCheck(End, 'END expected');
    RETURN n;
END ifStmt;

(* <elsif> ::= <expr> THEN <stmts> <else> *)
PROCEDURE elsif(routine:symbol):node;
VAR n1, n2:node;
BEGIN
    n1 := expr();
    boolCheck(n1);
    tokenErrorCheck(Then, 'THEN expected');
    n2 := stmts(routine);
    RETURN makeIfNode(n1, n2, else(routine));
END elsif;

(* <else> ::= <empty> | ELSIF <elsif> | ELSE <stmts>
   We can tell an <else> is empty by seeing if the next token is END. *)
PROCEDURE else(routine:symbol):node;
BEGIN
    CASE getTokenClass() OF
        End: ungetToken;
            RETURN emptyNode;
        Elsf: RETURN makeStmtsNode(elsif(routine), emptyNode);
        Else: RETURN stmts(routine);
    ELSE
        compError('END, ELSIF or ELSE expected');
        ungetToken;
        RETURN emptyNode;
    END;
END;

```


END else;

(* <while> ::= WHILE <expr> DO <stmts> END *)

PROCEDURE whileStmt(routine:symbol):node;

VAR n:node;

BEGIN

 n := expr();

 boolCheck(n);

 tokenErrorCheck(DO, 'DO expected');

 n := makeWhileNode(n, stmts(routine));

 tokenErrorCheck(End, 'END expected');

 RETURN n;

END whileStmt;

(* <return> ::= RETURN | RETURN <expr> *)

PROCEDURE returnStmt(routine:symbol):node;

BEGIN

 IF symbolEqual(routine, programName) THEN
 compError("can't return from main program");

 END;

 IF peekTokenClass() = Semicolon THEN

 RETURN makeReturnNode(routine, emptyNode);

 ELSE

 RETURN makeReturnNode(routine, expr());

 END;

END returnStmt;

(* We can't distinguish an assignment from a call based on the first token of the statement, since in both cases it's an identifier. The next token, though, will distinguish: it's an assignment sign for an assignment. *)

PROCEDURE assignOrCallStmt(t:token):node;

BEGIN

 IF getTokenClass() = Assignment THEN
 RETURN assignStmt(t);

 ELSE

 ungetToken;

 RETURN callStmt(t);

 END;

END assignOrCallStmt;

(* <assign> ::= <id> := <expr> *)

PROCEDURE assignStmt(varName:token):node;

VAR s:symbol;

BEGIN

 s := findSymbol(varName.string);

 IF NOT symbolClassEqual(s, Variable) THEN

 compError('only variables can be assigned to');

 RETURN expr(); (* consume the expression anyway *)

 ELSE

 RETURN makeAssignmentNode(s, expr());

 END;

END assignStmt;

(* <call> ::= <id> <actuals> *)

PROCEDURE callStmt(routineName:token):node;

VAR proc:symbol;

BEGIN

 proc := findSymbol(routineName.string);

 IF NOT symbolClassEqual(proc, Proc) THEN

 compError('only procedures can be used in a call statement');

 RETURN actuals();

 ELSE

 RETURN makeCallNode(proc, actuals());

 END;

END callStmt;

(* <actuals> ::= <empty> | (<exprlist>)

We can recognize an empty <actuals> by seeing if the next character is a left parenthesis. *)

PROCEDURE actuals():node;

VAR n:node;

BEGIN

 IF getTokenClass() = Lparen THEN

 n := exprlist();

 tokenErrorCheck(Rparen, 'right paren expected');

(continued)

```

    RETURN n;
ELSE
    ungetToken;
    RETURN emptyNode;
END;
END actuals;

(* <exprlist> ::= <expr> | <expr> , <exprlist>
   Exprlist always returns an nList node, even if there's only one expr. *)
PROCEDURE exprlist():node;
VAR n:node;
BEGIN
    n := expr();
    IF getTokenClass() = Comma THEN
        RETURN makeExprListNode(n, exprlist());
    ELSE
        ungetToken;
        RETURN makeExprListNode(n, emptyNode);
    END;
END exprlist;

BEGIN
END Parser.

```

```

+++++++
Start Routines.DEF
+++++++
DEFINITION MODULE Routines;

```

(* The part of the parser that deals with procedures and functions.

Syntax:

```

<routines> ::= <empty> | <proc> <routines> | <func> <routines>
<proc> ::= procedure <id> <formals> ; <vars> <block> ;
<func> ::= function <id> <formals> : <type> ; <vars> <block> ;
<formals> ::= <empty> | ( <formlist> )
<formlist> ::= <formdecl> | <formdecl> ; <formlist>
<formdecl> ::= <idlist> : <typeId>

```

*)

```
EXPORT QUALIFIED routines;
```

```
PROCEDURE routines;
```

```
END Routines.
```

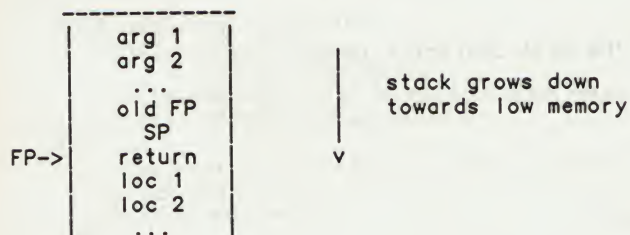
```

+++++++
Start Routines.MOD
+++++++
IMPLEMENTATION MODULE Routines;

```

(* The part of the parser that handles procedures and functions.

There are basically two things that have to be done: the routine declarations have to be processed to yield symbol table entries, and the code for the routine bodies has to be generated. The lists of formal parameters (arguments) and locals variables are placed in the appropriate slots in the symbol table entry for the routine. For functions, the return type of the function is put in the type slot of the symbol; for procedures, this slot is left undefined. An offset from the stack pointer is given to each local and formal. The initial offsets assume the following stack conventions:



The list of formals must be backwards to match the argument conventions.
The order of the locals doesn't matter, but it's also backwards.

We generate code as if for a block, with two exceptions: at the beginning, we have to push enough words to move the stack pointer past the local storage area; while we are at it, we initialize the words to 0. At the end, we generate a return, in case the user didn't. For procedures, it is okay to return by falling off the end. For functions, something has to be returned explicitly; it is an error to fall through.
*)

```
FROM Token IMPORT token, tokenClass, isType, typeType, tokenClassToType,
    tokenList, tIsEmpty, tIsNext, tIsToken, freeTokenList;
FROM LexAn IMPORT getToken, getTokenClass, ungetToken, tokenErrorCheck,
    getTokenErrorCheck, compError, peekTokenClass;
FROM Symbol IMPORT symbol, setSymbolType, setSymbolOffset,
    symbolFormals, symbolLocals, symbolList, sIsNext, sIsSymbol, sIsEmpty,
    SymbolClass, numFormals, numLocals;
FROM SymbolTable IMPORT enterSymbol, enterFormal, beginRoutine, endRoutine;
FROM CodeGen IMPORT genBlock, genLocals;
FROM CodeWrite IMPORT writeInt, writeFReturn, writeReturn, writeRoutineLabel;
FROM Parser IMPORT vars, idlist, block;
FROM Node IMPORT node;
```

```
CONST
    initialLocalOffset = -1;          (* offsets, in words, from the FP *)
    initialFormalOffset = 3;
```

```
(* <routines> ::= <empty> | <proc> <routines> | <func> <routines> *)
PROCEDURE routines;
```

```
BEGIN
    LOOP
        CASE getTokenClass() OF
            Procedure: proc;
            | Function: func;
            ELSE ungetToken; EXIT;
        END;
    END;
END routines;
```

```
(* <proc> ::= procedure <id> <formals> ; <vars> <routines> <block> ; *)
```

```
PROCEDURE proc;
VAR t:token;
    s:symbol;
    i:CARDINAL;
BEGIN
    getTokenErrorCheck(t, Identifier, 'procedure name expected');
    s := enterSymbol(t.string, Proc, tUnknown);
    beginRoutine(s);
    formals(s);
    tokenErrorCheck(Semicolon, 'semicolon expected');
    locals(s);
    routines;
    writeRoutineLabel(s);
    genLocals(s);
    genBlock(block(s));
    writeReturn(numFormals(s));
    tokenErrorCheck(Semicolon, 'semicolon expected');
    endRoutine(s);
END proc;
```

```
(* <func> ::= function <id> <formals>:<type>; <vars> <routines> <block>; *)
```

```
PROCEDURE func;
VAR fname, ftype:token;
    s:symbol;
    i:CARDINAL;
BEGIN
    getTokenErrorCheck(fname, Identifier, 'function name expected');
    s := enterSymbol(fname.string, Func, tUnknown);
    beginRoutine(s);
    formals(s);
    tokenErrorCheck(Colon, 'colon expected');
    getToken(ftype);
```

(continued)

```

    IF NOT isType(ftype.class) THEN
        compError('function type expected');
    END;
    tokenErrorCheck(Semicolon, 'semicolon expected');
    setSymbolType(s, tokenClassToType(ftype.class));
    locals(s);
    routines;
    writeRoutineLabel(s);
    genLocals(s);
    genBlock(block(s));
    (* Here we should generate an error message in the code: value not
       returned from function. But since we have no string manipulation,
       we can't. Instead we'll return either 0 (for an Integer function) or
       false (which is also 0) for a boolean function. *)
    writeInt(0);
    writeFReturn(numFormals(s));
    tokenErrorCheck(Semicolon, 'semicolon expected');
    endRoutine(s);
END func;

(* <formals> ::= <empty> | ( <formlist> ) *)
PROCEDURE formals(routine:symbol);
VAR formList:symbolList;
    offset:INTEGER;
BEGIN
    IF getTokenClass() = Lparen THEN
        formList(routine);
        tokenErrorCheck(Rparen, 'right paren expected');
    ELSE
        ungetToken;
    END;
    (* Set the offsets of the formals *)
    formList := symbolFormals(routine);
    offset := initialFormalOffset;
    WHILE NOT sIsEmpty(formList) DO
        setSymbolOffset(sSymbol(formList), offset);
        INC(offset);
        formList := sNext(formList);
    END;
END formals;

(* <formlist> ::= <formdecl> | <formdecl> ; <formlist> *)
PROCEDURE formlist(routine:symbol);
BEGIN
    formdecl(routine);
    IF getTokenClass() = Semicolon THEN
        formlist(routine);
    ELSE
        ungetToken;
    END;
END formlist;

(* <formdecl> ::= <idlist> : <typeId> *)
PROCEDURE formdecl(routine:symbol);
VAR tl, tokenp:tokenList;
    t:token;
    tt:typeType;
BEGIN
    tl := idlist();
    tokenErrorCheck(Colon, "colon expected");
    getToken(t);
    IF isType(t.class) THEN
        tt := tokenClassToType(t.class);
    ELSE
        compError("type name expected");
        tt := tUnknown;
        ungetToken;
    END;
    (* create and enter the symbols *)
    tokenp := tl;
    WHILE NOT tlEmpty(tokenp) DO
        tlToken(tokenp, t);
        enterFormal(t.string, tt, routine);
        tokenp := tlNext(tokenp);
    END;

```



```

    freeTokenList(tl);
END formdecl;

PROCEDURE locals(routine:symbol);
(* Syntactically, locals look just like globals; but we have to put them
   into the locals list of the routine and give them offsets from the frame
   pointer. *)
VAR locList:symbolList;
    offset:INTEGER;
BEGIN
    vars(routine);
    locList := symbolLocals(routine);
    offset := initialLocalOffset;
    (* set the offsets of the locals *)
    WHILE NOT sIsEmpty(locList) DO
        setSymbolOffset(sISymbol(locList), offset);
        DEC(offset);
        locList := sINext(locList);
    END;
END locals;

BEGIN
END Routines.

+++++++
Start Symbol.DEF
+++++++
DEFINITION MODULE Symbol;

(* The symbol data structure contains all the information about symbols (like
   variables and routine names). Symbol lists are used for lists of formals
   and locals. *)

FROM Token IMPORT stringType, tokenClass, typeType;

EXPORT QUALIFIED symbol, emptySymbol, SymbolClass,
    symbolClass, symbolString, symbolType, symbolLexLevel, symbolOffset,
    symbolFormals, symbolLocals, symbolNext, symbolPrev, symbolTokClass,
    setSymbolFormals, setSymbolLocals, setSymbolType, setSymbolNext,
    setSymbolPrev, setSymbolOffset, setSymbolTokClass,
    symbolClassEqual, sIsEmpty, symbolEqual,
    newSymbol, freeSymbol, numFormals, numLocals,
    symbolList, emptySymbolList, sIsEmpty, sISymbol, sINext, addToSymbolList,
    freeSymbolList;

TYPE
    symbol;
    symbolList;
    SymbolClass = (* the different kinds of symbols *)
        (Proc, Func, Variable, Keyword, Undeclared);

VAR emptySymbol:symbol;
    emptySymbolList:symbolList;

    (** Symbols **)

PROCEDURE symbolClass(s:symbol):SymbolClass;
(* Return the class of the symbol *)

PROCEDURE symbolString(s:symbol; VAR str:stringType);
(* Return the name of the symbol, as a string *)

(* Symbols are declared to be of a certain type (except procedures) *)
PROCEDURE symbolType(s:symbol):typeType;
PROCEDURE setSymbolType(s:symbol; tt:typeType);

PROCEDURE symbolLexLevel(s:symbol):CARDINAL;
(* Return the lexical level at which the symbol was declared *)

(* Each formal and local has an offset from the frame pointer. *)
PROCEDURE symbolOffset(s:symbol):INTEGER;
PROCEDURE setSymbolOffset(s:symbol; o:INTEGER);
(* These are for routines. They get and set the lists of formals and locals.
*)

```

(continued)

```

PROCEDURE symbolFormals(s:symbol):symbolList;
PROCEDURE symbolLocals(s:symbol):symbolList;
PROCEDURE setSymbolFormals(s:symbol; sl:symbolList);
PROCEDURE setSymbolLocals(s:symbol; sl:symbolList);

(* Return the number of formals or locals in the routine. *)
PROCEDURE numFormals(s:symbol):CARDINAL;
PROCEDURE numLocals(s:symbol):CARDINAL;

(* These next two are for implementing a doubly linked list. See
SymbolTable. *)
PROCEDURE symbolNext(s:symbol):symbol;
PROCEDURE symbolPrev(s:symbol):symbol;
PROCEDURE setSymbolNext(s1, s2:symbol);
PROCEDURE setSymbolPrev(s1, s2:symbol);

(* Keyword symbols have a corresponding token class. *)
PROCEDURE symbolTokClass(s:symbol):tokenClass;
PROCEDURE setSymbolTokClass(s:symbol; tc:tokenClass);

PROCEDURE symbolClassEqual(s:symbol; sc:SymbolClass):BOOLEAN;
(* Returns TRUE if the class of s equals sc. *)

PROCEDURE symbolEqual(s1, s2:symbol):BOOLEAN;
(* Returns TRUE if the two symbols are the same. *)

PROCEDURE symbolEmpty(s:symbol):BOOLEAN;
(* Returns TRUE if the symbol is the emptySymbol. *)

PROCEDURE newSymbol(VAR str:stringType; sc:SymbolClass; scop:CARDINAL;
                    tt:typeType):symbol;
(* Creates a new symbol. *)

PROCEDURE freeSymbol(s:symbol);
(* Frees the storage associated with s. *)

    (** Symbol Lists **)

PROCEDURE s1Empty(sl:symbolList):BOOLEAN;
(* Returns TRUE if sl is the empty symbol list. *)

PROCEDURE s1Next(sl:symbolList):symbolList;
(* Gets the rest of the symbol list. *)

PROCEDURE s1Symbol(sl:symbolList):symbol;
(* Gets the first symbol in the list *)

PROCEDURE addToSymbolList(s:symbol; sl:symbolList):symbolList;
(* Adds s to sl at the front. Return the new symbol list. *)

PROCEDURE freeSymbolList(sl:symbolList);
(* Frees the storage associate with sl (but NOT the storage of the symbols
in sl! *)

END Symbol.

+++++++
Start Symbol.MOD
+++++++
IMPLEMENTATION MODULE Symbol;

(* Symbol and symbol list data structures. *)

FROM Token IMPORT stringType, tokenClass, typeType, tokenClassToType;
FROM Storage IMPORT ALLOCATE, DEALLOCATE;
FROM MyTerminal IMPORT fatal;

TYPE
    symbol = POINTER TO symbolRec;
    symbolList = POINTER TO s1Rec;

    symbolRec = RECORD
        string: stringType;
        lexLevel: CARDINAL;
        type: typeType;
        next, prev: symbol;

```



```

        offset:INTEGER;
        CASE class: SymbolClass OF
            Proc, Func: formals, locals: symbolList;
            | Keyword:      tokClass: tokenClass;
        END;
    END;

    sIRec = RECORD
        sym: symbol;
        next: symbolList;
    END;

    (** getting fields **)

PROCEDURE symbolClass(s:symbol):SymbolClass;
BEGIN
    RETURN s^.class;
END symbolClass;

PROCEDURE symbolString(s:symbol; VAR str:stringType);
BEGIN
    str := s^.string;
END symbolString;

PROCEDURE symbolType(s:symbol):typeType;
BEGIN
    RETURN s^.type;
END symbolType;

PROCEDURE symbolLexLevel(s:symbol):CARDINAL;
BEGIN
    RETURN s^.lexLevel;
END symbolLexLevel;

PROCEDURE symbolOffset(s:symbol):INTEGER;
BEGIN
    RETURN s^.offset;
END symbolOffset;

PROCEDURE symbolFormals(s:symbol):symbolList;
BEGIN
    IF (s^.class = Proc) OR (s^.class = Func) THEN
        RETURN s^.formals;
    ELSE
        fatal('symbolFormals: not a proc or func');
    END;
END symbolFormals;

PROCEDURE symbolLocals(s:symbol):symbolList;
BEGIN
    IF (s^.class = Proc) OR (s^.class = Func) THEN
        RETURN s^.locals;
    ELSE
        fatal('symbolLocals: not a proc or func');
    END;
END symbolLocals;

PROCEDURE symbolNext(s:symbol):symbol;
BEGIN
    IF s = emptySymbol THEN
        fatal('symbolNext: empty symbol given');
    ELSE
        RETURN s^.next;
    END;
END symbolNext;

PROCEDURE symbolPrev(s:symbol):symbol;
BEGIN
    RETURN s^.prev;
END symbolPrev;

PROCEDURE symbolTokClass(s:symbol):tokenClass;
BEGIN
    IF s^.class = Keyword THEN
        RETURN s^.tokClass;

```

(continued)

```

ELSE
    fatal('symbolTokClass: not a keyword');
END;
END symbolTokClass;

    (** setting fields **)

PROCEDURE setSymbolFormals(s:symbol; sl:symbolList);
BEGIN
    IF (s^.class = Proc) OR (s^.class = Func) THEN
        s^.formals := sl;
    ELSE
        fatal('setSymbolFormals: not a proc or func');
    END;
END setSymbolFormals;

PROCEDURE setSymbolLocals(s:symbol; sl:symbolList);
BEGIN
    IF (s^.class = Proc) OR (s^.class = Func) THEN
        s^.locals := sl;
    ELSE
        fatal('setSymbolLocals: not a proc or func');
    END;
END setSymbolLocals;

PROCEDURE setSymbolType(s:symbol; tt:typeType);
BEGIN
    s^.type := tt;
END setSymbolType;

PROCEDURE setSymbolNext(s1, s2:symbol);
BEGIN
    s1^.next := s2;
END setSymbolNext;

PROCEDURE setSymbolPrev(s1, s2:symbol);
BEGIN
    s1^.prev := s2;
END setSymbolPrev;

PROCEDURE setSymbolOffset(s:symbol; o:INTEGER);
BEGIN
    s^.offset := o;
END setSymbolOffset;

PROCEDURE setSymbolTokClass(s:symbol; tc:tokenClass);
BEGIN
    IF s^.class = Keyword THEN
        s^.tokClass := tc;
    ELSE
        fatal('setSymbolTokClass: not a keyword');
    END;
END setSymbolTokClass;

(** other symbol procedures **)

PROCEDURE symbolClassEqual(s:symbol; sc:SymbolClass):BOOLEAN;
BEGIN
    RETURN (s^.class = Undeclared) OR (s^.class = sc);
END symbolClassEqual;

PROCEDURE symbolEqual(s1, s2:symbol):BOOLEAN;
BEGIN
    RETURN s1 = s2;
END symbolEqual;

PROCEDURE symbolEmpty(s:symbol):BOOLEAN;
BEGIN
    RETURN s = emptySymbol;
END symbolEmpty;

PROCEDURE newSymbol(VAR str:stringType; sc:SymbolClass; ll:CARDINAL;
    tt:typeType):symbol;
VAR s:symbol;
BEGIN

```



```

NEW(s); (* should be: NEW(s, sc); *)
WITH s^ DO
  string := str;
  lexLevel := ll;
  type := tt;
  next := emptySymbol;
  prev := emptySymbol;
  class := sc;
  CASE class OF
    Proc, Func:
      formals := emptySymbolList;
      locals := emptySymbolList;
    | Variable: offset := 0;
  ELSE (* do nothing *)
  END;
END;
RETURN s;
END newSymbol;

PROCEDURE freeSymbol(s:symbol);
BEGIN
  DISPOSE(s); (* should be: DISPOSE(s, s^.class); *)
END freeSymbol;

PROCEDURE numFormals(s:symbol):CARDINAL;
VAR formList:symbolList;
    count:CARDINAL;
BEGIN
  count := 0;
  formList := symbolFormals(s);
  WHILE NOT sIsEmpty(formList) DO
    INC(count);
    formList := sNext(formList);
  END;
  RETURN count;
END numFormals;

PROCEDURE numLocals(s:symbol):CARDINAL;
VAR locList:symbolList;
    count:CARDINAL;
BEGIN
  count := 0;
  locList := symbolLocals(s);
  WHILE NOT sIsEmpty(locList) DO
    INC(count);
    locList := sNext(locList);
  END;
  RETURN count;
END numLocals;

      (** symbolList **)

PROCEDURE sIsEmpty(sl:symbolList):BOOLEAN;
BEGIN
  RETURN sl = emptySymbolList;
END sIsEmpty;

PROCEDURE sNext(sl:symbolList):symbolList;
BEGIN
  RETURN sl^.next;
END sNext;

PROCEDURE sSymbol(sl:symbolList):symbol;
BEGIN
  RETURN sl^.sym;
END sSymbol;

PROCEDURE addToSymbolList(s:symbol; sl:symbolList):symbolList;
VAR newsl: symbolList;
BEGIN
  NEW(newsl);
  newsl^.sym := s;
  newsl^.next := sl;
  RETURN newsl;
END addToSymbolList;

```

(continued)

```

PROCEDURE freeSymbolList(sl:symbolList);
BEGIN
  IF NOT slEmpty(sl) THEN
    freeSymbolList(slNext(sl));
    DISPOSE(sl);
  END;
END freeSymbolList;

```

```

BEGIN
  emptySymbol := NIL;
  emptySymbolList := NIL;
END Symbol.

```

```

+++++++
Start SymbolTable.DEF
+++++++

```

```

DEFINITION MODULE SymbolTable;

```

```

(* The symbol table associates symbol records with names of symbols. *)

```

```

FROM Symbol IMPORT symbol, SymbolClass;
FROM Token IMPORT stringType, tokenClass, typeType;

```

```

EXPORT QUALIFIED enterSymbol, enterLocal, enterFormal, findSymbol,
findKeyword,
enterKeyword, beginRoutine, endRoutine, currentLexLevel;

```

```

PROCEDURE currentLexLevel():CARDINAL;
(* Returns the current lexical level *)

```

```

(* Enter global, local, formal, keyword symbols into the table.
enterSymbol is the general routine and returns the entered symbol. If the
symbol is already present, an error is signalled. EnterLocal and
enterFormal are used for local variables and formal parameters only; they
take care of inserting the symbol into the list of locals or formals,
respectively, which is associated with the routine. *)

```

```

PROCEDURE enterSymbol(VAR s:stringType; symc:SymbolClass; tt:
typeType):symbol;
PROCEDURE enterLocal(VAR s:stringType; tt:typeType; routine:symbol);
PROCEDURE enterFormal(VAR s:stringType; tt:typeType; routine:symbol);
PROCEDURE enterKeyword(s:stringType; tc:tokenClass);

```

```

PROCEDURE findSymbol(VAR s:stringType):symbol;
(* Look up the symbol in the table and return it. Return the empty symbol
if not found. *)

```

```

PROCEDURE findKeyword(VAR s:stringType; VAR tc:tokenClass):BOOLEAN;
(* Look up the keyword in the table and put its corresponding token class
in tc. Return FALSE if the symbol wasn't found. *)

```

```

PROCEDURE beginRoutine(rname:symbol);
(* To be called just after the routine name has been entered. Increments
lexical level. Also assigns a unique number to the routine if it isn't
global. *)

```

```

PROCEDURE endRoutine(rname:symbol);
(* Clean up the symbol table after a routine has been compiled. This includes
deleting the locals and formals from the table. *)

```

```

END SymbolTable.

```

```

+++++++
Start SymbolTable.MOD
+++++++

```

```

IMPLEMENTATION MODULE SymbolTable;

```

```

(* The symbol table for the SIMPL compiler. It is a hash table; each entry
is a symbol, possibly linked through the NEXT field to other symbols. The
list of symbols is doubly linked, to make it easy to delete from the
middle.

```

```

We still have to rehash to delete from the beginning, though. This could
be gotten around by hanging a dummy record off of every hashtable entry.
*)

```

```

FROM Symbol IMPORT symbol, SymbolClass, emptySymbol, newSymbol,
symbolFormals, symbolLocals, symbolEmpty, symbolString, symbolLexLevel,

```



```

symbolNext, symbolPrev, setSymbolNext, setSymbolPrev, setSymbolLocals,
setSymbolFormals, freeSymbol, symbolClass, symbolEqual, symbolTokClass,
setSymbolTokClass, emptySymbolList, numFormals, setSymbolOffset,
symbolList, sIsEmpty, sNext, sSymbol, addToSymbolList, freeSymbolList;
FROM Token IMPORT stringType, tokenClass, typeType;
FROM LexAn IMPORT compError;
FROM MyTerminal IMPORT fatal;
FROM StringStuff IMPORT stringEqual;

CONST symTabSize = 20;
(* This is NOT an upper limit on the number of symbols,
since we have linked lists coming off of the hashtable entries. Still,
the compiler may run faster (because the lists it searches are shorter)
if this number is increased. *)

VAR symbolTable: ARRAY[0..symTabSize-1] OF symbol;
    lexicalLevel: CARDINAL;

PROCEDURE currentLexLevel():CARDINAL;
BEGIN
    RETURN lexicalLevel;
END currentLexLevel;

PROCEDURE enterLocal(VAR s:stringType; tt:typeType; routine:symbol);
VAR sym:symbol;
BEGIN
    sym := enterSymbol(s, Variable, tt);
    setSymbolLocals(routine, addToSymbolList(sym, symbolLocals(routine)));
END enterLocal;

PROCEDURE enterFormal(VAR s:stringType; tt:typeType; routine:symbol);
VAR sym:symbol;
BEGIN
    sym := enterSymbol(s, Variable, tt);
    setSymbolFormals(routine, addToSymbolList(sym, symbolFormals(routine)));
END enterFormal;

PROCEDURE enterKeyword(s:stringType; tc:tokenClass);
VAR sym:symbol;
BEGIN
    sym := enterSymbol(s, Keyword, tUnknown);
    setSymbolTokClass(sym, tc);
END enterKeyword;

    (** symbol insertion **)

PROCEDURE enterSymbol(VAR s:stringType; symc:SymbolClass; tt:
typeType):symbol;
(* This does the real work of entering a symbol. It signals an error
if a symbol is redefined. *)
VAR sym:symbol;
    h:CARDINAL;
BEGIN
    sym := lookup(s, FALSE, h);
    IF symbolEmpty(sym) THEN
        RETURN insert(newSymbol(s, symc, lexicalLevel, tt), h);
    ELSE
        compError('redefined symbol');
        RETURN sym;
    END;
END enterSymbol;

    (** symbol lookup **)

PROCEDURE findSymbol(VAR s:stringType):symbol;
VAR sym: symbol;
    h: CARDINAL;
BEGIN
    sym := lookup(s, TRUE, h);
    IF symbolEmpty(sym) THEN
        compError('undefined symbol');
        RETURN insert(newSymbol(s, Undeclared, 0, tUnknown), h);
    ELSE
        RETURN sym;
    END;
END findSymbol;

```

```

PROCEDURE findKeyword(VAR s:stringType; VAR tc:tokenClass):BOOLEAN;
(* This is used by the lexical analyzer to return the keyword's token class.
Returns true if the keyword is found; tc will then contain the token
class of the keyword. *)
VAR sym:symbol;
    h:CARDINAL;
BEGIN
    sym := lookup(s, TRUE, h);
    IF symbolEmpty(sym) OR (symbolClass(sym) <> Keyword) THEN
        RETURN FALSE;
    ELSE
        tc := symbolTokClass(sym);
        RETURN TRUE;
    END;
END findKeyword;

PROCEDURE lookup(VAR s:stringType; anything:BOOLEAN; VAR h:CARDINAL):symbol;
(* Looks up the string in the symbol table. Returns the empty symbol if
the string isn't found; if it is, returns the symbol and, in h, the hash
value. anything TRUE means: "match anything".
This is what findSymbol uses. We match lexical level on insertion, to
check for redefined symbols.
*)
VAR sym: symbol;
    syms: stringType;
BEGIN
    h := hash(s);
    sym := symbolTable[h];
    WHILE NOT symbolEmpty(sym) DO
        symbolString(sym, syms);
        IF stringEqual(syms, s) AND
            (anything OR (lexicalLevel = symbolLexLevel(sym))) THEN
            RETURN sym;
        END;
        sym := symbolNext(sym);
    END;
    RETURN emptySymbol;
END lookup;

PROCEDURE insert(s:symbol; h:CARDINAL):symbol;
(* Link the symbol into the h'th symbol table entry. The symbol is put at
the front of the list. *)
BEGIN
    setSymbolNext(s, symbolTable[h]);
    setSymbolPrev(s, emptySymbol);
    symbolTable[h] := s;
    RETURN s;
END insert;

MODULE begRout; (* This needs to be a module because a variable needs
to be remembered across invocations. *)
IMPORT symbolLexLevel, setSymbolOffset, symbol, lexicalLevel;
EXPORT beginRoutine;

    VAR num:INTEGER;

    PROCEDURE beginRoutine(rname:symbol);
    BEGIN
        IF symbolLexLevel(rname) <> 0 THEN (* assign a unique number to *)
            setSymbolOffset(rname, num); (* non-global procedures *)
            INC(num);
        END;
        INC(lexicalLevel);
    END beginRoutine;

BEGIN
    num := 0;
END begRout;

PROCEDURE endRoutine(rname:symbol);
(* This is the stuff we do at the end of compiling a procedure or function.
The free's are just to reclaim storage. The remove's remove the symbols
from the symbol table, which is important if some local symbol is
shadowing a global symbol. We remove both locals and formals, but we don't
free the formals because we need them for type checking.
We also remove the routines declared at this lexical level, and free
their

```


formals. We find these routines by searching the entire symbol table--it would probably be better to keep a list of them.

```

*)
BEGIN
  removeSymbolList(symbolLocals(rname));
  freeSymbols(symbolLocals(rname));
  freeSymbolList(symbolLocals(rname));
  setSymbolLocals(rname, emptySymbolList);
  removeSymbolList(symbolFormals(rname));
  removeRoutinesAtThisLevel;
  DEC(lexicalLevel);
END endRoutine;

PROCEDURE removeSymbolList(symbolp:symbolList);
BEGIN
  WHILE NOT s!Empty(symbolp) DO
    removeSymbol(s!Symbol(symbolp));
    symbolp := s!Next(symbolp);
  END;
END removeSymbolList;

PROCEDURE removeRoutinesAtThisLevel;
(* Remove all routines defined at this lexical level. Free their formals.
   All the symbols at this lexical level will be at the beginning of their
   respective buckets in the symbol table, and they all will be routines. *)
VAR i:CARDINAL;
    s, next:symbol;
BEGIN
  FOR i := 0 TO symTabSize-1 DO
    s := symbolTable[i];
    WHILE (NOT symbolEmpty(s)) AND (symbolLexLevel(s) = lexicalLevel) DO
      freeSymbols(symbolFormals(s));
      freeSymbolList(symbolFormals(s));
      (* remove this symbol from the table *)
      next := symbolNext(s);
      symbolTable[i] := next;
      IF NOT symbolEmpty(next) THEN
        setSymbolPrev(next, emptySymbol);
      END;
      freeSymbol(s);
      s := next;
    END;
  END;
END removeRoutinesAtThisLevel;

PROCEDURE removeSymbol(s:symbol);
(* Splice the symbol out of the symbol table. If the symbol is at the
   beginning of the list, we have to rehash to find the right entry.
   Otherwise, just remove it from the list. *)
VAR bucket:CARDINAL;
    syms: stringType;
BEGIN
  IF symbolEmpty(symbolPrev(s)) THEN
    symbolString(s, syms);
    bucket := hash(syms);
    IF NOT symbolEqual(symbolTable[bucket], s) THEN
      fatal('removeSymbol: error');
    ELSE
      symbolTable[bucket] := symbolNext(s);
      IF NOT symbolEmpty(symbolNext(s)) THEN
        setSymbolPrev(symbolNext(s), emptySymbol);
      END;
    END;
  ELSE
    setSymbolNext(symbolPrev(s), symbolNext(s));
    IF NOT symbolEmpty(symbolNext(s)) THEN
      setSymbolPrev(symbolNext(s), symbolPrev(s));
    END;
  END;
END removeSymbol;

PROCEDURE freeSymbols(symbolp:symbolList);
VAR nextSymbol:symbolList;
BEGIN
  WHILE NOT s!Empty(symbolp) DO

```

(continued)

January

```
        nextSymbol := sNext(symbolp);
        freeSymbol(sNext(symbolp));
        symbolp := nextSymbol;
    END;
END freeSymbols;
```

(*** low-level stuff ***)

```
PROCEDURE hash(VAR s:stringType):CARDINAL;
(* A simple hash function: just add up the ASCII values of the characters. *)
VAR i, sum:CARDINAL;
BEGIN
    i := 0;
    sum := 0;
    WHILE s[i] <> 0C DO
        sum := sum + ORD(s[i]);
        INC(i);
    END;
    RETURN sum MOD symTabSize;
END hash;
```

```
PROCEDURE initSymbolTable;
VAR i:CARDINAL;
BEGIN
    FOR i := 0 TO symTabSize-1 DO
        symbolTable[i] := emptySymbol;
    END;
END initSymbolTable;
```

```
BEGIN
    initSymbolTable;
    lexicalLevel := 0;
END SymbolTable.
```

```
+++++++
Start Token.DEF
+++++++
```

DEFINITION MODULE Token;

```
(* Tokens are what the lexical analyzer returns to the parser. Keywords are
distinct tokens, as are the special characters like parens, colon, etc.
This module also exports typeType, which is a list of the possible types
of variables in SIMPL. For now, SIMPL only has integers, booleans and
characters. TokenLists are lists of tokens; they are used in the "varlist"
procedure of the parser.
*)
```

```
EXPORT QUALIFIED token, tokenClass, stringType, stringlen, isType, isRelation,
typeType, tokenClassToType,
tokenList, emptyTokenList, tlToken, tlNext, addToTokenList,
tlEmpty, freeTokenList;
```

CONST stringlen = 80;

TYPE

```
tokenClass = (And, Assignment, Begin, Boolean, Char, Character, Colon,
Comma, Divide, Do, Else, Elself, End, EndOfInput, Equal, False,
Function, Greater, GreaterEqual, Identifier, If, Int, Integer, Less,
LessEqual, Lparen, Minus, Not, NotEqual, Or, Period, Plus, Procedure,
Program, Read, Return, Rparen, Semicolon, String, Then, Times, True,
UMinus, Var, While, Write);
```

stringType = ARRAY[0..stringlen] OF CHAR;

```
token = RECORD
    CASE class:tokenClass OF
        Identifier, String: string: stringType;
        | Int: integer: INTEGER;
        | Character: ch: CHAR;
    END;
END;
```



```

tokenList;

typeType = (tInteger, tBoolean, tChar, tUnknown);

VAR emptyTokenList: tokenList;

PROCEDURE tokenClassToType(tc:tokenClass):typeType;
(* Converts the token class Integer to the type tInteger, and so on. *)

PROCEDURE isType(tc:tokenClass):BOOLEAN;
(* Returns TRUE if tc = Integer, Char, or Boolean *)

PROCEDURE isRelation(tc:tokenClass):BOOLEAN;
(* Returns TRUE if tc is a relational operator (Equal, Greater, etc.) *)

PROCEDURE tToken(tl:tokenList; VAR t:token);
(* Gets the first token in the token list. *)

PROCEDURE tNext(tl:tokenList):tokenList;
(* Gets the rest of the token list. *)

PROCEDURE addToTokenList(VAR t:token; tl:tokenList):tokenList;
(* Add a token to the beginning of the token list. *)

PROCEDURE freeTokenList(tl:tokenList);
(* Free the storage used by the token list. *)

PROCEDURE tIsEmpty(tl:tokenList):BOOLEAN;
(* Return TRUE if the token list is empty. *)

END Token.

+++++++
Start Token.MOD
+++++++
IMPLEMENTATION MODULE Token;

(* Tokens and token lists for the SIMPL compiler. *)

FROM Storage IMPORT ALLOCATE, DEALLOCATE;
FROM Terminal IMPORT WriteString;

TYPE tokenList = POINTER TO tokenListRec; (* token lists are linked lists *)
    tokenListRec = RECORD
        tok: token;
        next: tokenList;
    END;

PROCEDURE tokenClassToType(tc:tokenClass):typeType;
BEGIN
    CASE tc OF
        Integer: RETURN tInteger;
        Boolean: RETURN tBoolean;
        Char: RETURN tChar;
    ELSE
        WriteString('tokenClassToType: unknown type');
        RETURN tUnknown;
    END;
END tokenClassToType;

PROCEDURE isType(tc:tokenClass):BOOLEAN;
BEGIN
    RETURN (tc = Integer) OR (tc = Boolean) OR (tc = Char);
END isType;

PROCEDURE isRelation(tc:tokenClass):BOOLEAN;
BEGIN
    RETURN (tc = Equal) OR (tc = NotEqual) OR (tc = Greater) OR
        (tc = GreaterEqual) OR (tc = Less) OR (tc = LessEqual);
END isRelation;

PROCEDURE tToken(tl:tokenList; VAR t:token);
BEGIN
    IF tIsEmpty(tl) THEN
        WriteString("tToken: empty tokenList");
    ELSE

```

(continued)

January

```
        t := tl^.tok;
    END;
END tlToken;

PROCEDURE tlNext(tl:tokenList):tokenList;
BEGIN
    RETURN tl^.next;
END tlNext;

PROCEDURE addToTokenList(VAR t:token; tl:tokenList):tokenList;
(* Create a token list record for the new token and splice it on to the
   front of the token list. Return ( a pointer to) the new record. *)
VAR newtl: tokenList;
BEGIN
    NEW(newtl);
    newtl^.tok := t;
    newtl^.next := tl;
    RETURN newtl;
END addToTokenList;

PROCEDURE freeTokenList(tl:tokenList);
BEGIN
    IF NOT tlEmpty(tl) THEN
        freeTokenList(tlNext(tl));
        DISPOSE(tl);
    END;
END freeTokenList;

PROCEDURE tlEmpty(tl:tokenList):BOOLEAN;
BEGIN
    RETURN tl = emptyTokenList;
END tlEmpty;

BEGIN
    emptyTokenList := NIL;
END Token.

+++++++
Start TypeChecker.DEF
+++++++
DEFINITION MODULE TypeChecker;

(* Handles the actual type-checking of SIMPL expressions and statements. *)

FROM Token IMPORT tokenClass, typeType;
FROM Node IMPORT node;
FROM Symbol IMPORT symbol;

EXPORT QUALIFIED typeCompatible, opAppropriate, callCheck, readCheck,
    writeCheck, boolCheck, assignCheck, binopCheck, unopCheck,
    returnCheck;

PROCEDURE typeCompatible(t1, t2:typeType):BOOLEAN;
(* Returns TRUE if t1 and t2 are compatible types. In order to avoid
   cascades of error messages, if one or both of the types is tUnknown,
   it still returns TRUE. *)

PROCEDURE opAppropriate(op:tokenClass; arg:node):BOOLEAN;
(* Returns TRUE if the type of the argument can be handled by the operator *)

PROCEDURE callCheck(routine:symbol; args:node);
(* Checks the procedure or function call for right number and types of args.
   *)

PROCEDURE readCheck(actuals:node);
(* Checks the call to the READ built-in procedure. *)

PROCEDURE writeCheck(actuals:node);
(* Checks the call to the WRITE built-in procedure. *)

PROCEDURE boolCheck(n:node);

PROCEDURE assignCheck(var:symbol; expr:node);

PROCEDURE returnCheck(routine:symbol; expr:node):BOOLEAN;
```



```

PROCEDURE binopCheck(op:tokenClass; leftarg, rightarg:node):BOOLEAN;

PROCEDURE unopCheck(op:tokenClass; arg:node):BOOLEAN;

END TypeChecker.

++++++
Start TypeChecker.MOD
++++++

IMPLEMENTATION MODULE TypeChecker;

(* Handles type-checking of SIMPL expressions. *)

FROM Node IMPORT node, nodeType, nodeFirst, nodeRest, nodeEmpty, nodeClass,
    NodeClass, nodeSymbol;
FROM Token IMPORT tokenClass, stringType, typeType;
FROM Symbol IMPORT symbol, symbolType, symbolFormals, symbolList,
    symbolString,
    s1Next, s1Symbol, s1Empty, symbolClassEqual, SymbolClass, symbolClass,
    numFormals;
FROM MyTerminal IMPORT fatal;
FROM LexAn IMPORT compError;

PROCEDURE opAppropriate(op:tokenClass; arg:node):BOOLEAN;
BEGIN
    CASE op OF
        Plus, Minus, UMinus, Times, Divide:
            RETURN typeCompatible(nodeType(arg), tInteger);
        | Greater, GreaterEqual, Less, LessEqual:
            RETURN typeCompatible(nodeType(arg), tInteger) OR
                typeCompatible(nodeType(arg), tChar);
        | And, Or, Not:
            RETURN typeCompatible(nodeType(arg), tBoolean);
        | Equal, NotEqual:
            RETURN TRUE;
        ELSE
            fatal("opAppropriate: unknown op type");
    END;
END opAppropriate;

PROCEDURE typeCompatible(t1, t2:typeType):BOOLEAN;
BEGIN
    IF (t1 = tUnknown) OR (t2 = tUnknown) THEN
        RETURN TRUE;
    ELSE
        RETURN t1 = t2;
    END;
END typeCompatible;

PROCEDURE callCheck(routine:symbol; args:node);
(* Tricky because formals are stored backwards in symbol, but forwards
   in the call to the routine. We do nothing if the symbol is not a procedure
   or function; that check is handled in the parser. *)
VAR nFormals, nActuals:CARDINAL;
    dummy:node;
BEGIN
    IF (symbolClass(routine) = Proc) OR (symbolClass(routine) = Func) THEN
        nFormals := numFormals(routine);
        nActuals := numActuals(args);
        IF nActuals < nFormals THEN
            compError('too few arguments to routine');
        ELSIF nActuals > nFormals THEN
            compError('too many arguments to routine');
        END;
        dummy := argsMatch(symbolFormals(routine), args);
    END;
END callCheck;

PROCEDURE argsMatch(flist:symbolList; alist:node):node;
(* This procedure matches two lists, one of which is backwards. It does
   it by recursing down one list all the way, then iterating down the other
   list while unrecursing. *)
BEGIN
    IF s1Empty(flist) THEN
        RETURN alist;
    
```

(continued)

```

ELSE
    alist := argsMatch(s1Next(flist), alist);
    IF nodeEmpty(alist) THEN
        RETURN alist;
    ELSE
        argCheck(s1Symbol(flist), nodeFirst(alist));
        RETURN nodeRest(alist);
    END;
END;
END argsMatch;

PROCEDURE argCheck(formal:symbol; actual:node);
VAR s:stringType;
BEGIN
    IF NOT typeCompatible(symbolType(formal), nodeType(actual)) THEN
        compError('type of formal does not match type of actual');
    END;
END argCheck;

PROCEDURE numActuals(actuals:node):CARDINAL;
VAR count:CARDINAL;
BEGIN
    count := 0;
    WHILE NOT nodeEmpty(actuals) DO
        INC(count);
        actuals := nodeRest(actuals);
    END;
    RETURN count;
END numActuals;

PROCEDURE readCheck(actuals:node);
VAR arg:node;
BEGIN
    IF nodeEmpty(actuals) THEN
        compError('READ requires an argument');
    ELSE
        REPEAT
            arg := nodeFirst(actuals);
            IF nodeClass(arg) <> nSymbol THEN
                compError('READ must have a variable as an argument');
            ELSIF NOT symbolClassEqual(nodeSymbol(arg), Variable) THEN
                compError('READ must have a variable as an argument');
            ELSIF NOT (typeCompatible(nodeType(arg), tInteger) OR
                typeCompatible(nodeType(arg), tChar)) THEN
                compError('READ can only read integers or characters');
            END;
            actuals := nodeRest(actuals);
        UNTIL nodeEmpty(actuals);
    END;
END readCheck;

PROCEDURE writeCheck(actuals:node);
VAR arg:node;
BEGIN
    IF nodeEmpty(actuals) THEN
        compError('WRITE requires an argument');
    ELSE
        REPEAT
            arg := nodeFirst(actuals);
            IF NOT (typeCompatible(nodeType(arg), tInteger) OR
                typeCompatible(nodeType(arg), tChar)) THEN
                compError('WRITE can only write integers or characters');
            END;
            actuals := nodeRest(actuals);
        UNTIL nodeEmpty(actuals);
    END;
END writeCheck;

PROCEDURE binopCheck(op:tokenClass; leftarg, rightarg:node):BOOLEAN;
BEGIN
    IF NOT opAppropriate(op, leftarg) THEN
        compError('inappropriate arg type: left arg');
        RETURN FALSE;
    END;
    IF NOT opAppropriate(op, rightarg) THEN
        compError('inappropriate arg type: right arg');
    END;
END binopCheck;

```



```

    RETURN FALSE;
END;
IF NOT typeCompatible(nodeType(leftarg), nodeType(rightarg)) THEN
    compError('argument types not compatible');
    RETURN FALSE;
ELSE
    RETURN TRUE;
END;
END binopCheck;

PROCEDURE unopCheck(op:tokenClass; arg:node):BOOLEAN;
BEGIN
    IF NOT opAppropriate(op, arg) THEN
        compError('inappropriate arg type');
        RETURN FALSE;
    ELSE
        RETURN TRUE;
    END;
END unopCheck;

PROCEDURE assignCheck(var:symbol; expr:node);
BEGIN
    IF NOT typeCompatible(symbolType(var), nodeType(expr)) THEN
        compError('types not assignment compatible');
    END;
END assignCheck;

PROCEDURE boolCheck(n:node);
BEGIN
    IF NOT typeCompatible(nodeType(n), tBoolean) THEN
        compError('Boolean expression expected');
    END;
END boolCheck;

PROCEDURE returnCheck(routine:symbol; expr:node):BOOLEAN;
BEGIN
    IF (NOT nodeEmpty(expr)) AND (symbolClass(routine) <> Func) THEN
        compError('only functions can return values');
        RETURN FALSE;
    ELSIF nodeEmpty(expr) AND (symbolClass(routine) <> Proc) THEN
        compError('function must return a value');
        RETURN FALSE;
    ELSIF (NOT nodeEmpty(expr)) AND
        (NOT typeCompatible(symbolType(routine), nodeType(expr))) THEN
        compError('return type not compatible with function type');
        RETURN FALSE;
    ELSE
        RETURN TRUE;
    END;
END returnCheck;

BEGIN
END TypeChecker.

```

prolog.doc

TEXT

"AI in Computer Vision" John L Cuadrado and Clara Y. Cuadrado.
 January, page 237. Prolog.doc references Pdprolog.exe, which is
 available in the FromBYTE85 file area on BIX (PC/MS-DOS only).

A.D.A PROLOG Documentation Version 1.7f for the Educational and Public
 Domain Versions October 28, 1985 Copyright Robert Morein and Automata Design
 Associates 1570 Arran Way Dresher, Pa. 19025 (215)-646-4894 News Release 1.7f
 fixes the last bugs (heh). ANYONE who has ever purchased a copy of any
 version of A.D.A. PROLOG is entitled to a no-charge update. Simply send us
 your original disk or a photocopy of the receipt. We have included in pre-
 release form Simon Blackwell's magnificent expert system shell. This system
 is tailored to A.D.A. PROLOG and includes forward and backward chaining, and
 a linkage method of insuring database consistency. Bayesian reasoning, rule
 editing, and documentation are forthcoming. The system was developed on VML
 PROLOG, but it is our intention to expeditiously insure that it runs under PD

(continued)

PROLOG as well. To that end, we have added I/O redirection, since that is how Simon stores things to disk. Copyright Notice The public domain PD PROLOG system has been contributed to the public domain for unrestricted use with one exception: the object code may not be disassembled or modified. Electronic bulletin boards and SIG groups are urged to aid in giving this software the widest possible distribution. This documentation may be reproduced freely, but it may not be included in any other documentation without the permission of the author. Introduction We are pleased to present the third major version of PD PROLOG, version 1.7. Version 1.7 continues to refine "problems" and adds the entertaining feature of IBM PC video screen support. The memory requirements are somewhat greater than the original, since it uses the large memory model. It compensates in thoroughness. The memory requirement is about 210K bytes of TPA, and it will benefit from up to 253k bytes. The available workspace is 100K bytes. We hope that you'll get some fun out of this PROLOG. It will afford you exposure to THE fifth generation language at the cost only of some intellectual effort. The motive is perfectly explicable: We want you to think of Automata Design Associates for fifth generation software. It also gives us a nice warm feeling. The memory requirement is 200 k of transient program area, plus whatever space is needed to execute programs from within PROLOG. DOS or MSDOS 2.0 are required. The program does not require IBM PC compatibility to run, although the screen access routines do require compatibility. Products by Automata Design Associates Automata Design Associates specializes in software for artificial intelligence and robotic applications. A PROLOG language system is available in various configurations. A LISP interpreter will be introduced in March of 1985. There are five versions of PROLOG available from Automata Design Associates. All of them run under the MSDOS or PCDOS operating systems. Other environments will be supported soon. Public Domain PROLOG This serves to further the general awareness of the public about PROLOG. It also is an excellent adjunct to anyone learning the language. Most of the core PROLOG described by Clocksin and Mellish in the book Programming In PROLOG is implemented. A complete IBM PC video screen support library is included in this and all other A.D.A. prologs. Trace predicates are not. This version is available from us for \$10.00 postage paid. Educational PROLOG At extremely modest cost this affords an educational institution or individual a PROLOG system which provides the maximum available programming area available within the 8086 small programming model. Tracing, a debugging aid, allows monitoring a program as it runs. User settable spy points selectively allow this. Exhaustive tracing is also available. I/O redirection gives some file ability. An "exec" function allows the execution of a program or editor from within PROLOG, thus encouraging an interactive environment. An "interrupt" menu is added, permitting the control of tracing, toggling the printer, and screen printing. Definite clause grammar support is now included. The cost of Educational PROLOG is \$29.95. FSM PROLOG A small increment in price adds full random access file capability. Character and structure I/O are allowed. The "asserta and "assertz" predicates are expanded and work with a clause indexing ability. One can assert clauses anywhere in the database under precise pattern matching control. A tree structured lexical scoping system and floating point arithmetic are other enhancements. The cost of FSM PROLOG is \$49.95. VMI PROLOG -- Virtual Memory (Replaces type VMS) At reasonable cost the addition of virtual memory gives an expansion of capabilities of an order of magnitude. The database on disk is treated transparently. No special provisions need be made by the user. Virtual and resident databases may be mixed. A unique updating algorithm preserves the format of the database as typed by the user while making only those changes necessary to make it equivalent to the database in central memory. The cost of VMI PROLOG is \$99.95. VML PROLOG Large model Virtual Memory System A.D.A. PROLOG is a remarkable fifth generation development tool for the implementation of intelligent strategies and optimized control. It is both the kernel for applications of virtually unlimited scope and a sophisticated development tool that multiplies the productivity of the programmer many times. With a cost/performance ratio exceeding that of any other product and a compatibility insured by compliance to the Edinburgh syntax, performance is enhanced by numerous extensions, many of them invisible to the user. A quick overview of some of the features discloses: 1) Invisible compilation to a semantic network preserves the flexibility of the interpreted mode and the speed of a compiler. The programmer can compile and recompile any portion of a module at any time. The edit/compile/test cycle is short and free of strain. An interface is provided to an editor of choice. 2) Floating point arithmetic with a full complement of input and output methods, transcendental and conversion functions. 3) Virtual memory. Module size and number is unrestricted. Resident and virtual modules may be coresident. Compilation is incremental. The cache algorithm is sophisticated. Changes made in the database can be updated to disk by a single command. 4) A powerful exec function, and acceptance of stream input make integration into applications practical. 5) Many additional built-in predicates enhance the efficiency of

the system. 6) Debugging facilities let you see your program run without any additional generation steps. 7) Totally invisible and incremental garbage collection. There is NEVER any wait for this function. The cost of this system is \$200 for the MSDOS version. Upgrade Policy Half the cost of any A.D.A. PROLOG interpreter may be credited to the purchase of a higher level version. The full cost of VMS prolog may be applied to the purchase of VMI or VML PROLOG. Updates to a particular level product vary from \$15.00 to \$35.00. Run-time Packages Software developers wishing to integrate an A.D.A. product into their system should inquire about specialized run-time packages available at reasonable cost. Technical Information Technical information may be obtained at (215) - 646- 4894 Perhaps we can answer the following questions in advance: There is no support for: APPLE II, Atari, Commodore, or CPM 80. Other machines from these manufactures may be supported in the future. The MSDOS products are available on 5" and 8" diskettes. To Place Your Order: You may place your order at the following number: (215)-646-4894 - day and night. Returns The software may be returned within 30 days of purchase for a full refund. This applies whether or not the software has been used. We do ask that manuals, disks and packaging be returned in excellent condition. How to run the Demonstration Programs without Knowing What You're Doing We strongly advise that you purchase the book Programming in PROLOG by Clocksin and Mellish, publisher Springer Verlag, 1981. For the impatient we give some advice. Type the demonstration program you wish to run. There must be at least one entry point within the program. Note: Please understand that these are demonstrations programs. Regarding user interface, they are poorly written. You will probably have to read Clocksin and Mellish to appreciate that the following examples of input are not equivalent: "yes.", "yes". The animals program - "animal" Most of the examples require C & M for comprehension. The program "animals", however, can be appreciated by anyone. It is a traditional example of an expert system. We had hoped to include the animals program on disk, but we have found to our dismay that the version which we used is allegedly copyrighted by the implementors of PROLOG 86. Don't be surprised - even "happy birthday" is copyrighted. We will simply point out that the November '84 issue of Robotics Age included a version of the animals game, which you can, at the risk of copyright infringement, type in. There is only one change that need be made. The "concat" function used in that program has arguments of the form: concat([atom1, atom2,...], result). In order to make the concat definition more closely resemble that of "name", which is described by Clocksin and Mellish, the arguments have been reversed: concat(result, [atom1, atom2,...]) Assuming that you have typed in the program and made the change just noted, the following steps are required to run it: Run the prolog.exe file. The prompt "?-" will appear. Type "consult('animals').<CR>". Here <CR> indicates you are to type a carriage return. The PROLOG system will load "animals" and compile it into an internal form. When the "?-" prompt appears PROLOG is ready to run the "animals" guessing game. The object of the program is to deduce the animal you are thinking of. To start it off type "help.<CR>". PROLOG will respond by asking a question. Because of the way the animals program is written, you must respond in a rigid format. You may type "yes<CR>", "no<CR>", or "why<CR>". Eventually the program will terminate with either a guess as to what animal you are thinking of, or a remark that the animal is not within its domain of knowledge. The program has learned, however. You may run the program again to see what effect additional knowledge has on the program's behavior. The program fragment "console" shows how you may improve the console input routines of any of these programs. The Hematology Diagnosis Program - "hemat" Although the logical structure is not as sophisticated as that of "animals", it is interesting for several reasons: 1) The program evaluates numerical data to arrive at a diagnosis. 2) Although inaccurate, it demonstrates that useful question answering systems are not difficult to write in PROLOG. 3) There are some mistakes in the program, which only slightly impede its usefulness. This program uses structure input. Terminate all your answers with a period, as in "y.<CR>", or "no.<CR>". The starting point is "signs.<CR>". PROLOG will prompt you for signs of anemia. The program attempts to diagnose two varieties of a hemolytic anemia. The program could use a good working over by a hematologist and we would be delighted to collaborate. Prime Number Generator - "sieve" This program demonstrates that anything can be programmed in PROLOG if one tries hard enough. Asking the question "primes(50, L).<CR>" causes a list of prime numbers less than 50 to be printed out. "Sieve" is heavily recursive and quickly exhausts the stack space of the small model interpreters. Grrrules This is an example of the use of the definite clause grammar notation. PD PROLOG does not have this translation facility, but ED PROLOG and all of our other versions do. It is possible to perform the translations by hand if you have thoroughly read C & M. Then you would have the pleasure of asking: ?-sentence(X, [every,man,loves,a,woman], []). and having the meaning elucidated as a statement in the predicate calculus. Special Offer # 1 For some inexplicable reason, demonstration

(continued)

programs are hard to come by. We are too busy writing PROLOG fill this gap. We will reward the contribution of "cute" sample programs with the following: 1) A free copy of type VMS virtual memory PROLOG 2) The sample program will be published as an intact file together with whatever comments or advertisements the author may see fit to include, on our distribution disks. 3) Exceptional contributions may merit a copy of type VML large model virtual memory PROLOG which now incorporates a UNIX1 style tree structured domain system. Special Offer # 2 If you are a hardware manufacturer and would like a PROLOG language for your system, the solution is simple. Just send us one of your machines! Provided your system implements a "C" compiler, it will be ported in no time flat. _____ 1. Trademark of AT & T. Writing Programs For ED PROLOG You do not type in programs at the "?-" prompt. There is no built-in editor. The command "consult(user)" is accepted but does not cause PROLOG to enter an editing mode. We feel that the most universally acceptable editing method is for the user to use a text editor of choice, which can be invoked from within PROLOG by the "exec" predicate. Use Wordstar or your customary editor to write a program. Then run PD PROLOG and use the consult function to load the program. In all cases except PD PROLOG, you can run your editor without leaving PROLOG by use of the "exec" predicate. Running the Interpreter COMMANDS: Give commands in lower case. TO RUN: Invoke PROLOG.EXE. After the "?-" prompt appears, type "consult(<filename><CR>)", where <filename> is the desired database. To exit, type "exitsys.<CR>" TO ASK A QUESTION: At the prompt, type "<expression>.<CR>", where <expression> is a question as described by Clocksin and Mellish. Be sure to terminate the question with a period. The question may be up to 500 characters long. TO INPUT A STRUCTURE AT THE KEYBOARD: The structure may be up to 500 characters in length. Be sure to terminate with a period. TO ASK FOR ANOTHER SOLUTION: If a solution has been provided, the PROLOG interpreter will ask "More? (Y/N):". Only if a "y" is typed will the interpreter perform a search. TO ABORT A SEARCH: Simply type the escape key. The interpreter will respond with "Interrupted.", and return to the command prompt. TO LOAD ANOTHER DATABASE: Type "consult(<filename>).<CR>" The file name must have the extension ".PRO". It is not necessary to include the extension in the argument of consult. The file name as given must not be the same as a predicate name in the file or any file which will be loaded. TO REMOVE A DATABASE: Type "forget(<filename>).<CR>" TO EXIT TO THE OPERATING SYSTEM: Type "exitsys.<CR>" The system is totally interactive; any commands the operator gives are and must be valid program statements. Statements must terminate with a period. All commands which take a file name also accept a path name. Any name which is not a valid PROLOG atom (refer to C & M) must be enclosed in single quotes. Thus one could say consult(expert) but one would need single quotes with consult('b:\samples\subtype\expert'). To exit the system, type "exitsys.<CR>" Atoms may contain MSDOS pathnames if they are enclosed by single quotes, i.e., 'b:\samples\animal'. You may consult more than one file at a time. However, all names are public and name conflicts must be avoided. The order in which modules are loaded may, in cases of poor program design, affect the behavior. Command Line Arguments ED PROLOG accepts one command line argument, which is the name of a "stream" which replaces the console for input. The "stream" in MSDOS is a pipe or file which supplies input until end-of-file is reached. Control then reverts back to the console. To avoid noisy parser error messages when end-of-file is reached, the last command in the file should be "see(user)." A Reference of Note With minor exceptions, the syntax is a superset of that described by Clocksin and Mellish in the book Programming in Prolog by W.F. Clocksin and C.S. Mellish, published by Springer Verlag in Berlin, Heidelberg, New York, and Tokyo. We shall refer to this book as "C & M". There are very few syntactical differences, mostly unrecognized and/or minor. When an operator is declared using the "op" statement, the operator must be enclosed in single quotes in the "op" statement itself, if it would not otherwise be a legal Edinburgh functor. Subsequently, however, the parser will recognize it for what it is, except in the "unop" statement, where it must again be enclosed in single quotes. Variable numbers of functor parameters are supported. A goal may be represented by a variable, which is less restrictive than the C & M requirement that all goals be functors. The variable must be instantiated to a functor when that goal is pursued. Rules which appear inside other expressions must be enclosed in parenthesis if the "," operator is to be recognized as a logical connective. All infix operators described by C & M, and user defined infix, prefix, and postfix operators with variable associativity and precedence are supported exactly as in C & M. The Built In Predicate Library Available Operators in PD and ED PROLOG Column 1 gives the function symbol. Column 2 gives the precedence. The range of precedence is 1 to 255. A zero in the precedence column indicates the symbol is parsed as a functor, and precedence is meaningless in this case. Column 3 gives the associativity. A zero in the associativity column indicates the symbol is parsed as a functor, and associativity is meaningless in this case. Column 4 indicates which version the function is available in. Unless otherwise noted,

the function is available in all versions. Nonstandard predicates are indicated by "NS". op/pred precedence associativity availability "!" 0 0 "|" 0 0 "=" 40, XFX "==" 40, XFX "\\=" 40, XFX "\\==" 40, XFX "/" 21, YFX "@=" 40, XFX ">=" 40, XFX "<=" 40, XFX ">" 40, XFX "<" 40, XFX "-" 31, YFX "*" 21, YFX "+" 31, YFX "=-" 40, XFX "-->" 255, YFY (not in PD PROLOG) "?-" 255, FY "arg" 0, 0, "asserta" 0, 0, "assertz" 0, 0, "atom" 0, 0, "atomic" 0, 0, "clause" 0, 0, "clearops" 0, 0, "cls" 0, 0, NS "concat" 0, 0, "consult" 8, FX, "crtgmode" 0, 0, NS "crtset" 0, 0, NS "curset" 0, 0, NS "curwh" 0, 0, NS "debugging" 0, 0, "dir" 0, 0, "display" 0, 0, "dotcolor" 0, 0, NS "drawchar" 0, 0, NS "drawdot" 0, 0, NS "drawline" 0, 0, NS "exec" 0, 0, "exitsys" 0, 0, NS "forget" 0, 0, NS "functor" 0, 0, "get0" 8, FX, "integer" 0, 0, "is" 40, XFX, "listing" 0, 0, "mod" 11, XFX, "name" 0, 0, "nl" 0, 0, "nodebug" 0, 0, (not in PD PROLOG) "nonvar" 0, 0, "nospy" 50, FX, (not in PD PROLOG) "not" 60 FX "notrace" 0, 0, (not in PD PROLOG) "op" 0, 0, "popoff" 0, 0, NS "popoffd" 0, 0, NS "popon" 0, 0, NS "popond" 0, 0, NS "print" 0, 0, "prtscr" 0, 0, NS "put" 0, 0, "ratom" 0, 0, "read" 0, 0, "recon" 0, 0, (Note: this is "reconsult") "repeat" 0, 0, "retract" 0, 0, "rnum" 0, 0, "see" 0, 0, (not in PD PROLOG) "seeing" 0, 0, (not in PD PROLOG) "seen" 0, 0, (not in PD PROLOG) "skip" 0, 0, (not in PD PROLOG) "spy" 50, FX, (not in PD PROLOG) "tab" 0, 0, "tell" 0, 0, (not in PD PROLOG) "telling" 0, 0, (not in PD PROLOG) "told" 0, 0, (not in PD PROLOG) "trace" 0, 0, (not in PD PROLOG) "true" 0, 0, "unop" 0, 0, "var" 0, 0, "write" 0, 0, Description of the Modifications. call(<goal>) The predicate as defined in C & M is obsolete. The purpose was to permit a goal search where the goal name was a variable instantiated to some functor name. A.D.A. permits writing of goals with such names, so the mediation of the "call" clause is no longer necessary. The "call" predicate may be trivially implemented for compatibility with the PROLOG definition call(X) :- X. clause The function clause(X, Y) has the new optional form clause(X, Y, I). If the third variable is written, it is instantiated to the current address of a clause in memory. The only use of the result is with succeeding assertfa and assertfz statements. debugging "Debugging" prints a list of the current spyoints. After each name a sequence of numbers may appear, indicating the number of arguments that is a condition of the trace. The word "all" appears if the number of arguments is not a condition of the trace. op(<prec>, <assoc>, <functor>) Defines the user definable grammar of a functor. The definition conforms to that in C & M. We mention here a minor but important point. If <functor> is not a valid PROLOG atom it must be enclosed in single quotes when declared in the "op" declaration. It is not necessary or legal to do this when the functor is actually being used as an operator. In version 1.6, a declared or built-in operator can be used either as an operator or as a functor. For example, +(2,3) = 2 + 3. is a true statement. Declared operators are annotated in the directory display with their precedence and associativity. Output predicates display write print put These functions have been modified to accept multiple arguments in the form: print(<arg1>, <arg2>, <arg3>, ...) Thus, "put(a, b, c)" would result in the display of "abc". The names of some PROLOG atoms that may occur are not accepted by the PROLOG scanner unless surrounded by single quotes. This only applies when such an atom is read in, not when it is internally generated. Nevertheless, this presents us with a problem: We would like to be capable of writing valid PROLOG terms to a file. In some cases, it is necessary to add the single quotes. In other cases, such as human oriented output, they are an irritant. The modified definitions of the following predicates are an attempt at a solution: display Operator precedence is ignored, all functors are printed prefix and single quotes are printed if needed or they were supplied if and when the atom was originally input. write Operator precedence is taken into account and operators are printed according to precedence. Single quotes are printed under the same conditions as for "display." print Operator precedence is taken into account and operators are printed according to precedence. Single quotes are never displayed. This is a human oriented form of output and should never be used for writing of terms for the PROLOG scanner. get0 read The functions "get0" and "read" have been extended to support input from a stream other than the one currently selected by "see". To direct output to a file or other stream, an optional argument is used. For example, "get0(char, <file name>)" or "get0(char, user)" would cause input to come from <file name> or the console. If the file has not already been opened, "get0" will fail. Atoms enclosed by single quotes, eg. '\nthis is a new line' can contain the escape sequences '\n', '\r', '\t' and '\\'. If these atoms are printed by "display" or "write" they are printed just as they are. If they are printed by the "print" clause they are translated as follows: '\n' results in the printing of a carriage return and a line feed. '\r' results in the printing of a carriage return only. '\t' results in the printing of a tab character. '\\' allows the printing of a single quote within a quoted atom. The "portray" feature is not presently implemented. Description of the New Predicates clearops- Nullify the operator status of every operator in the

(continued)

database. concat((<variable> | <functor>), <List>) A list of functors or operators is concatenated into one string, which becomes the name of a new atom to which <variable> or <functor> must match or be instantiated. dir(option) Provide an alphabetized listing to the console of atoms, constants, or open files. Without options, simply type "dir.<CR>". Options are: dir(pred) - list clause names only. dir(files) - list open files only. exitsys Exit to the operating system. forget(<file name>) Make a database unavailable for use and reclaim the storage it occupied. ratom(<arg>, <stream>)- Read an atom from the input stream, to which <arg> matches or is instantiated. <stream> is optional. If <stream> is not given, the input stream defaults to the standard input. Input is terminated by a CR or LF, which are not included in the stream. Arithmetic Capabilities Integer arithmetic is supported. Numbers are 32 bit signed quantities. The following arithmetic operators are supported: "+", "-", "*", "/", "<", "<=", ">", ">=", mod. Arithmetic operators must never be used as goals, although they may be part of structures. It is legal to write: $X = a + b$ which results in the instantiation of X to the structure (a + b). But the following is not legal: alpha(X, Y) :- X + Y, beta(Y). Evaluation of an arithmetic expression is mediated by the "is" and inequality predicates. For instance, the following would be correct: alpha(X, Y, Z) :- Z is X + Y. beta(X, Y) :- X + 2 < Y + 3. IBM PC Video Display Predicates A high level method is provided for drawing and displaying on the screen of IBM PC and compatible computers. cls Clear the screen and position the cursor at the upper left hand corner. crtmode(X) Matches the argument to the mode byte of the display which is defined as follows: mode meaning 0 40 x 25 BW (default) 1 40 x 25 COLOR 2 80 x 25 BW 3 80 x 25 COLOR 4 320 x 200 COLOR 5 320 x 200 BW 6 640 x 200 BW 7 80 x 25 monochrome display card crtset(X) This sets the mode of the display. The argument must be one of the modes given above. curset(<row>, <column>, <page>) Sets the cursor to the given row, column, and page. The arguments must be integers. curwh(<row>, <column>) Reports the current position of the cursor. The argument must be an integer or variable. The format is: 1) page zero is assumed. 2) The row is in the range 0 to 79, left to right. 3) The column is in the range 0 to 24, bottom to top. dotcolor(<row>, <column>, <color>) The argument <color> is matched to the color of the specified dot. The monitor must be in graphics mode. drawchar(<character>, <attribute>) Put a character at the current cursor position with the specified attribute. The arguments <character> and <attribute> must be integers. Consult the IBM technical reference manual regarding attributes. drawdot(<row>, <column>, <color>) Put a dot at the specified position. The monitor must be in the graphics mode. The arguments must be integer. The argument <color> is mapped to integers by default in the following manner: drawline(<X1>, <Y1>, <X2>, <Y2>, <color>) Draw a line on the between the coordinate pairs. The monitor must be in the graphics mode and the arguments are integer. prtscr Print the screen as it currently appears. Be sure that the printer is on line and ready before invoking this predicate, since otherwise, the system may lock up or abort. The integer argument <color> referred to in the above predicates is represented as follows: COLOR PALETTE 0 PALETTE 1 0 background background 1 green cyan 2 red magenta 3 brown white To change the palette and the background, see the IBM Technical Reference Bios listings for more information. Trace Files (type ED only) You can now dump your trace to disk, instead of (groan) wasting reams of printer paper. This option is described in the next section. The Interrupt Menu (type ED only) This menu has been modified. It was formerly called the ESCAPE menu, but the meaning of the ESCAPE key has been redefined. It is no longer necessary to display the menu to perform one of the menu functions. This reduces the amount of display which is lost by scrolling off the screen. At any time while searching, printing, or accepting keyboard input, you can break to this menu. It is generally not possible to do this during disk access, since control passes to the operating system at this time. Two keys cause this break to occur: ^V: The menu is displayed and a command is accepted at the prompt "INTERRUPT>". After a command, the menu is redisplayed until the user selects a command which causes an exit. ^I: The menu is not displayed. Command is accepted at the prompt "INTERRUPT>" until the user selects a command which causes an exit. ESC: Typing this key causes a termination of the PROLOG search and control returns to the user command level with a prompt of "?-". Notice that previously, the ESC key invoked this menu. As the resulting menu indicates, the following functions are possible: A: Abort the search and return to the prompt. O Open a trace file. The user is prompted for the file name. The file receives all trace output. If a file is already opened it is closed with all output preserved. C Close the trace file if one is open. Otherwise there is no effect. ^C: Immediately exit PROLOG without closing files. This is not advised. ^P: Typing <Control>P toggles the printer. If the printer is on, all input and output will also be routed to the printer. S: If the machine in use is an IBM PC compatible machine, the currently displayed screen will be printed. If the machine is not an IBM PC compatible, do not use this function. T: If trace is in use, most of the trace output can be temporarily

turned off by use of this function, which is a toggle. R: Entering another ESC causes a return to the current activity (keyboard input or search) with no residual effect from the interruption. Conserving memory space Success popping is controlled by the predicates "popond", "popoffd", "popon", and "popoff". Success popping is means of reclaiming storage which is used on backtracking to reconstruct how a particular goal was satisfied. If it is obvious that there is no alternative solution to a goal this PROLOG system is smart enough to reclaim that storage. In this system, success popping is an expensive operation. Therefore, there is a tradeoff of memory versus time. On the other hand, discrete use of success popping can actually speed up a program by recreating structures in a more accessible form. The definitions of the control predicates is given in this manual and their use is totally optional. The modulation of success popping has no effect on program logic (read solution.) The "cut" can save substantial time and computational overhead as well as storage. Although the execution of the cut costs time, you can design your program to use cuts in critical places to avoid unnecessary backtracking. Thus the execution speed of the program can actually increase. Anyone who has read Clocksin and Mellish is aware, of course, that the "cut" has a powerful logical impact which is not always desirable. popoff See the below definition. popon The inference engine does complete success popping for goals which appear after "popon". Consider this example: goal :- a, popon, b, c, popoff, d. If no alternative solutions exist for b, then success popping will reclaim storage by removing unnecessary records describing how "b" was satisfied. If the Prolog system cannot rule out possible additional solutions, success popping will never occur, regardless of your use of "popon". Since goal "d" occurs after "popoff", success popping will never occur. popoffd If no "popon" or "popoff" declarations occur in a clause, the default action is determined by "popoffd" and "popond". If "popoffd" has been invoked, the default is that success popping will not occur. popond The inverse of "popoffd". Turns on default success popping. printf(<stream>, <term1>, <term2>, ...) Prolog Tutorial Introduction Probably you have heard of the language PROLOG within the last year or so. You probably wondered the following things: 1) What does the name stand for? Names of computer languages are almost always acronyms. 2) What is it good for? 3) Why now? 4) Can I get a copy to play with? Congratulations! You obviously know the answer to the fourth question. We now respond to the other three. 1) The name stands for "programming in logic." This we shall elaborate on in depth later on. 2) PROLOG is good for writing question answering systems. It is also good for writing programs that perform complicated strategies that compute the best or worst way to accomplish a task, or avoid an undesirable result. 3) PROLOG was virtually unknown in this country until researchers in Japan announced that it was to be the core language of that country's fifth generation computer project. This is the project with which Japan hopes to achieve a dominant position in the world information industry of the 1990's. PROLOG is one of the most unusual computer languages ever invented. It cannot be compared to FORTRAN, PASCAL, "C", or BASIC. The facilities complement, rather than replace those of conventional languages. Although it has great potential for database work, it has nothing in common with the database languages used on microcomputers. Perhaps the best point to make is that while conventional languages are prescriptive, PROLOG is descriptive. A statement in a conventional language might read: if(car_wheels = TRUE) then begin (some sort of procedure) X = X + 1; end A statement in PROLOG could just be a statement of fact about cars and wheels. There are many relationships that hold. For instance, has(car, wheels). has(car, quant(wheels, four)). round(wheels). Each of these statements is an independent fact relating cars, wheels, and the characteristics of wheels. Because they are independent, they can be put into a PROLOG program by programmers working separately. The man who is a specialist on car bodies can say his thing, the wheel specialist can have his say, and the participants can work with relative independence. And this brings to light a major advantage of PROLOG: PARALLEL PROGRAMMING!!! With conventional programming languages projects can still be "chunked", or divided between programmers. But efficiency on a team project drops drastically below that of the individual programmer wrapped up in his own trance. As the number of participants grows the need for communication grows geometrically. The time spent communicating can exceed that spent programming! Although PROLOG does not eliminate the need for task coordination, the problem is considerably simplified. It also provides the ability to answer questions in a "ready to eat form." Consider your favorite BASIC interpreter. Based upon the statements about cars and wheels previously given, could you ask it the following question? has(car, X), round(X). Does a car have anything which is round? The question instructs the PROLOG interpreter to consider all the objects that it knows are possessed by a car and find those which are round. Perhaps you are beginning to guess that PROLOG has the abilities of a smart database searcher. It can not only find

(continued)

the facts but selectively find them and interpret them. Consider the problem of a fault tree, as exemplified by this abbreviated one: {Car won't start} | | [Engine turns over](No) --> [Battery voltage](no)-\ (Yes) v | {Check battery} | [Smell gasoline](yes) --> {Try full throttle cranking} | (failure) /-----/ | (details omitted) The fault tree is easily programmed in BASIC. Later we shall show that PROLOG supplies a superior replacement for the fault tree. Though the tree is capable of diagnosing only the problem for which it was designed, PROLOG dynamically constructs the appropriate tree from facts and rules you have provided. PROLOG is not limited to answering one specific question. Given enough information, it will attempt to find all deductive solutions to any problem. PROLOG PRIMER I. Rules and Facts This is where you should start if you know nothing about PROLOG. Let us consider a simple statement in PROLOG, such as: 1) has(car, wheels). This statement is a "fact." The word "has" in this statement is known either as a functor or predicate. It is a name for the relationship within the parenthesis. It implies that a car has wheels. But the order of the words inside the bracket is arbitrary and established by you. You could just as easily say: 2) has(wheels, car). and if you wrote this way consistently, all would be well. The words has, wheels, and car are all PROLOG atoms. "Wheels" and "car" are constants. A database of facts can illustrate the data retrieval capabilities of PROLOG. For instance: 3) has(car, wheels). has(car, frame). has(car, windshield). has(car, engine). You could then ask PROLOG the question: 4) has(car, Part). The capital "P" of Part means that Part is a variable. PROLOG will make Part equal to whatever constant is required to make the question match one of the facts in the database. Thus PROLOG will respond: Part = wheels. More?(Y/N): If you type "y" the next answer will appear: Part = frame. More?(Y/N): If you continue, PROLOG will produce the answers Part = windshield and Part = engine. Finally, you will see: More?(Y/N):y No. indicating that PROLOG has exhausted the database. Incidentally, when a variable is set equal to a constant or other variable, it is said to be instantiated to that object. Notice that PROLOG searches the database forwards and in this case, from the beginning. The forward search path is built into PROLOG and cannot be changed. An author of a program written in a prescriptive language is quite conscious of the order of execution of his program, while in PROLOG it is not directly under his control. The other major element is the rule which is a fact which is conditionally true. In logic this is called a Horn clause: 5) has(X, wheels) :- iscar(X). The fact iscar(car) and the above rule are equivalent to 6) has(car, wheels). The symbol :- is the "rule sign." The expression on the left of :- is the "head" and on the right is the body. The variable X has scope of the rule, which means that it has meaning only within the rule. For instance, we could have two rules in the database using identically named variables. 7) has(X, transportation) :- has(X, car), has(license, X). 8) has(X, elephant) :- istrainer(X), hasjob(X). The variables X in the two expressions are completely distinct and have nothing to do with each other. The comma between has(X, car) and has(license, X) means "and" or logical conjunction. The rule will not be true unless both the clauses has(X, car) and has(license, X) are true. On the other hand if there is a rule 9) has(license, X) :- passedexam(X). consider what PROLOG will do in response to the question: 10) has(harry, transportation). (Notice that harry has not been capitalized because we do not want it taken as a variable. We could, however, say 'Harry' enclosed in single quotes.) It will scan the database and use (7), in which X will be instantiated to harry. The rule generates two new questions: 11) has(harry, car). 12) has(license, harry). Assuming that harry has a car, the first clause of (7) is satisfied and the database is scanned for a match to (12). PROLOG picks up rule (9) in which X is instantiated to harry and the question is now posed: 13) passedexam(harry). If there is a fact: passedexam(harry). in the database then all is well and harry has transportation. If there is not, then PROLOG will succinctly tell you: No. But suppose Harry has money and can hire a chauffeur as any good programmer can. That could be made part of the program in the following way. The rule which PROLOG tried to use was: 14) has(X, transportation) :- has(X, car), has(license, X). At any point following it there could be included another rule: 15) has(X, transportation) :- has(X, money). or simply the bald fact: 16) has(harry, transportation). These additional rules or facts would be used in two circumstances. If at any point a rule does not yield a solution, PROLOG will scan forward from that rule to find another applicable one. This process is known as "backtracking search" and can be quite time consuming. If in response to the "More?" prompt you answer "y" PROLOG will search for an additional distinct solution. It will attempt to find an alternate rule or fact for the last rule or fact used. If that fails, it will back up to the antecedent rule and try to find an alternate antecedent. And it will continue to back up until it arrives at the question you asked, at which point it will say: No. "Antecedent" to a rule means that it gave rise to its' use. For example, (7) is the antecedent of (9) in the context of the question (16). II. Grammar It is a boring subject, but it must be discussed. All PROLOG statements are composed of valid terms, possibly a rule sign (":-

"), commas representing conjunction ("and"), and a period at the very end. A term is a structure, constant, variable, or number. What is a structure? It is a kind of grouping: 1) Structures consist of a functor, and a set of objects or structures in parenthesis. 2) Objects are constants, variables, numbers, or lists, which we have not discussed yet. 3) A constant or functor must be a string of letters and numbers, beginning with a lower case letter, unless you choose to enclose it in single quotes ('howdy pardner'), in which case you are freed from these restrictions. 4) A variable must be a string of letters and numbers beginning with a capital letter. 5) A functor may optionally have arguments enclosed in parenthesis, as in: hascar(X) or hascar. An example: "has(X, transportation)." is a structure. III. Input / Output You now know enough to write simple databases and interrogate them profitably. But before we examine more sophisticated examples, it will be necessary to add input and output to the language. There are built in functions which appear as rules which are satisfied once. Thus the statement: write('Hello world.'). can be included on the right side of a rule: greetings(X) :- ishuman(X), write('Hello world.'). You can also write "write(X)" where X is some variable. Note that 'Hello world.' is not enclosed in double quotes. Single quotes, which denote a constant, are required. Double quotes would denote a list, which is another thing entirely. Provided that a match to "ishuman" can be found, the builtin function "write" is executed and the message printed to the screen. The builtin read(X) reads a "structure" that you input from the keyboard. More formally, we have read(<variable> or <constant>) write(<variable> or <constant>) If you write read(Input), then the variable "keyboard" will be assigned to whatever is typed at the keyboard, provided that the input is a valid PROLOG structure. The builtin "read" will fail if instead of Keyboard we wrote read(baloney), where "baloney" is a constant, and the user at the keyboard did not type exactly "baloney." When you input a structure in response to a "read" statement, be sure to end it with a period and an <ENTER>. There is a convenient way of putting the cursor on a new line. This is the builtin "nl". For example: showme :- write('line 1'), nl, write('line 2'). would result in: line 1 line 2 There is also a primitive form of input/output for single characters. It will be discussed later. IV. A Fault Tree Example Consider the "won't start" fault tree for an automobile: {Car won't start} | | [Engine turns over](No) --> [Battery voltage](no)-\ (Yes) v | [Check battery] | [Smell gasoline](yes) --> {Try full throttle cranking} | (failure) /-----/ | | /-----/ | | | | [Check for fuel line leaks](yes)--> {Replace fuel line} | (no) | | | | [Check for defective carburetor](yes)--\ | (no) v | [Repair carburetor] \----\ | | [Is spark present](no)--> [Do points open and close](no)-\ | (yes) v /----/ | [Adjust points] | /-----/ | | [Pull distributor wire, observe spark](blue)--\ | (orange) v | | [Check plug wires & cap] | | | [Measure voltage on coil primary](not 12V)--\ | (12V) v | | [Check wiring, ballast resistor] | | | [Check condenser with ohmmeter](conducts)--\ | (no conduction) v | | [Replace condenser] | | | [Open and close points](voltage not 0 - 12)--\ | (voltage swings 0 - 12) v | | [Fix primary circuit] | | | [Consider hidden fault, swap components] | | \-----{Call a tow truck!!} A PROLOG program to implement this is simple. Each statement represents a decision point fragment of the tree. The PROLOG interpreter dynamically assembles the tree as it attempts a solution. 'car wont start' :- write('Is the battery voltage low?'), affirm, nl, write('Check battery'). 'car wont start' :- write('Smell gasoline?'), affirm, nl, 'fuel system' :- write('Try full throttle cranking'). 'fuel system' :- write('Are there fuel line leaks?'), affirm, nl, write('Replace fuel line.'). 'fuel system' :- write('Check carburetor'). 'car wont start' :- write('Is spark present?'), not(affirm), nl, 'no spark' :- write('Do points open and close?'), not(affirm), nl, write('Adjust or replace points.'). 'no spark' :- write('Is the spark off the coil good?'), affirm, write('Check plug wires and cap.'). 'no spark' :- write('What is the voltage on the primary of the coil:'), read(Volts), Volts < 10, nl, write('Check wiring and ballast resistor.'). 'no spark' :- write('Does the capacitor leak?'), affirm, write('Replace the capacitor.'). 'no spark' :- not('primary circuit'). 'primary circuit' :- write('Open the points. Voltage across coil?:'), nl, read(Openvolts), Openvolts < 1, write('Close the points. Voltage across coil?:'), read(Closevolts), Closevolts > 10, nl, write('Primary circuit is OK.'). 'no spark' :- write('Consider a hidden fault. Swap cap, rotor, points, capacitor.'). 'Car wont start' :- write('Get a tow truck!!'). --End program-- The above is a simple example of an expert system. A sophisticated system would tell you exactly the method by which it has reached a conclusion. It would communicate by a "shell" program written in PROLOG which would accept a wider range of input than the "valid structure" required by the PROLOG interpreter directly. V. Lists Consider a shopping list given you by your wife. It is a piece of paper with items written on it in an order that probably symbolizes their importance. At the top it may say

(continued)

EGGS!!!, followed by carrots, hamburger, and finally a flea collar for the dog, if you can find one. In PROLOG such a list would be written: 1) [eggs, carrots, hamburger, fleacollar] The order of a list is important so that eggs and carrots cannot be reversed and PROLOG be uncaring. Let us put the list in a structure: shopping([eggs, carrots, hamburger, fleacollar]). Then if you wished to isolate the head of the list you could ask the question: shopping([Mostimportant | Rest]). and PROLOG would respond: Mostimportant = eggs, Rest = [carrots, hamburger, fleacollar]. The vertical bar "|" is crucial here. It is the string extraction operator, which performs a combination of the CDR and CAR functions of LISP. When it appears in the context [X|Y] it can separate the head of the list from the rest, or tail. You may have gained the impression that PROLOG is a rather static language capable of answering simple questions, but it is far more powerful than that. The string extraction operator is the key. It permits PROLOG to whittle a complex expression down to the bare remainder. If the rules you have given it permit it to whittle the remainder down to nothing, then success is achieved. An example of this is the definition of "append." Let us suppose you have not yet done yesterday's shopping, let alone today's. You pull it out of your wallet and sootch tape it to the list your wife just gave you. Yesterday's list was: [tomatoes, onions, ketchup] Combined with [eggs, carrots, hamburger, fleacollar] we

obtain[eggs,carrots,hamburger,fleacollar,tomatoes,onions,garlic]. To take one list and to attach it to the tail of another list is to "append" the first to the second. The PROLOG definition of append is: Rule1: append([], L, L). Rule2: append([X|List1], List2, [X|List3]) :- append(List1, List2, List3]. The general scheme is this: The definition consists of one rule and one fact. The rule will be used over and over again until what little is left matches the fact. The [] stands for empty list, which is like a bag without anything in it. This is an example of a recursive definition. Suppose we ask: append([a,b,c], [d,e,f], Whatgives). 1. Rule 2 is invoked with arguments ([a,b,c], [d,e,f], Whatgives). 2. Rule 2 is invoked again with arguments: ([b,c], [d,e,f], List3). 3. Rule 2 is invoked again with arguments: ([b], [d,e,f], List3). 4. The arguments are now ([], [d,e,f], List3). Rule 1 now matches. End. How does this cause a list to be constructed? The key is to watch the third argument. Supplied by the user, it is named "Whatgives". The inference engine matches it to [X|List3] in rule 2. Now lets trace this as rule two is successively invoked: Whatgives | | | v Rule2: [X|List3] (List1 = [b,c]) | \ | \ | \ v \ Rule2: a [X'|List3'] (List1' = [c]) | \ | \ | \ v \ Rule2: b [X''|List3''] (List1'' = [], ie., empty set.) | \ | \ | \ Rule1: c L (in Rule1 = [d,e,f]) End. L in rule 1 is [d,e,f] for the following reason: Notice that rule 2 never alters List2. It supplies it to whatever rule it invokes. So L in rule 1 is the original List2, or [a,b,c]. This example would not have worked if the order of rules one and two were reversed. The PROLOG inference engine always attempts to use the the first rule encountered. You could imagine it as always reading your program from the top down in attempting to find an appropriate rule. Since rule 2 would always satisfy, an unpleasant thing would have happened if the order were reversed. The program would loop forever. I hope that this tiny introduction to PROLOG whets your appetite. You should now purchase the book Programming In Prolog W.F. Clocksin and C.S. Mellish Springer - Verlag Berlin, Heidelberg, New York. 1981, 1984 Springer - Verlag

February

Linetest.pas
TEXT
"Turbo Pascal 3.0" Mark Bridger.
February, page 281

```
program LINETEST;  
var  
  i, j : integer;  
begin  
  graphmode;  
  Palette(1);
```

```
    for i := 0 to 15 do  
      for j := 0 to 9 do  
        draw(20*i, 20*j, 319-20*i, 199-20*j, i+j)  
        write(chr(7)) {Beep}  
      end.  
end.
```

Btrans.pas
TEXT
"Turbo Pascal 3.0" Mark Bridger.
February, page 281

```
program BTRANS;  
var  
  F,G: file; {untyped files for blockmoves}  
  buffer: array[1..128] of byte;  
  I: integer;  
begin  
  assign(F, 'infile.dat');  
  assign(G, 'outfile.dat');  
  reset(F); rewrite(G);
```

```
    while not EOF(F) do  
      begin  
        blockread(F, buffer, 1,I);  
        blockwrite(G, buffer, 1,I)  
      end;  
    close(F); close(G);  
    write(chr(7)); {Beep}  
  end.
```

Trans.pas
TEXT
"Turbo Pascal 3.0" Mark Bridger.
February, page 281

```
program TRANS;  
var  
  F,G: file of byte;  
  ch: byte;  
begin  
  assign(F, 'infile.txt');  
  assign(G, 'outfile.txt');  
  reset(F); rewrite(G);
```

```
    while not EOF(F) do  
      begin  
        read(F, ch);  
        write(G, ch)  
      end;  
    close(F); close(G);  
    write(chr(7)); {Beep}  
  end.
```

Calc.pas
TEXT
"Turbo Pascal 3.0" Mark Bridger.
February, page 281

```
program CALC;  
var A,B,C: real;  
N, I: integer;  
begin  
  N:= 5000;  
  A:= 2.71828;  
  B:= 3.14159;  
  C:= 1;  
  For I:= 1 to N do
```

```
    begin  
      C:= C * A;  
      C:= C * B;  
      C:= C/A;  
      C:= C/B  
    end;  
    write(chr(7));  
    writeln('Error = ', C - 1)  
  end.
```

(continued)

Float.pas
TEXT
"Turbo Pascal 3.0" Mark Bridger.
February, page 281

```
program FLOAT;
var I: Integer;
    x,y: real;
begin
    x:= 1;
    for I:= 1 to 1000 do
        begin
            y:= sin(x);
```

```
        y:= ln(x);
        y:= exp(x);
        y:= sqrt(x);
        y:= arctan(x);
        x:= x + 0.01
    end
end.
```

Sieve.pas
TEXT
"Turbo Pascal 3.0" Mark Bridger.
February, page 281

```
program SIEVE;
const
    size = 8190;
var
    flags: array[0..size] of boolean;
    I, prime, K, count, iter: integer;
begin
    writeln('Start one iteration.');
```

```
    for iter:= 1 to 10 do
        begin
            count:= 0;
            for I:= 0 to size do
                flags[I]:= true;
            for I:= 0 to size do
                if flags[I]
```

```
            then
                begin
                    prime:= I + I + 3;
                    k:= I + prime;
                    while K <= size do
                        begin
                            flags[K]:= false;
                            K:= K + prime;
                        end;
                    count:= count + 1
                end;
            writeln(count:1, ' primes.');
```

```
        end;
    writeln(chr(7)) {Beep}
end.
```

Qsort.pas
TEXT
"Turbo Pascal 3.0" Mark Bridger.
February, page 281

```
Eprogram QuickSort(input,output);
const
    max = 100;
type
    standardArray = array[0..max] of real;
var
    numbers: standardArray;
    last: integer;
```

```
procedure swap(var a,b : real);
var
    t : real;
begin
    t := a;
    a := b;
    b := t;
end;
```

```
procedure printArray(top : integer);
var
    i : integer;
begin
    writeln('-MARK-');
end;
```

```
procedure getArray(var top:integer);
const
    worstCase : standardArray =
        (100.0, 99.0, 98.0, 97.0, 96.0, 95.0, 94.0, 93.0, 92.0, 91.0,
```



```

90.0, 89.0, 88.0, 87.0, 86.0, 85.0, 84.0, 83.0, 82.0, 81.0,
80.0, 79.0, 78.0, 77.0, 76.0, 75.0, 74.0, 73.0, 72.0, 71.0,
70.0, 69.0, 68.0, 67.0, 66.0, 65.0, 64.0, 63.0, 62.0, 61.0,
60.0, 59.0, 58.0, 57.0, 56.0, 55.0, 54.0, 53.0, 52.0, 51.0,
50.0, 49.0, 48.0, 47.0, 46.0, 45.0, 44.0, 43.0, 42.0, 41.0,
40.0, 39.0, 38.0, 37.0, 36.0, 35.0, 34.0, 33.0, 32.0, 31.0,
30.0, 29.0, 28.0, 27.0, 26.0, 25.0, 24.0, 23.0, 22.0, 21.0,
20.0, 19.0, 18.0, 17.0, 16.0, 15.0, 14.0, 13.0, 12.0, 11.0,
10.0, 9.0, 8.0, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0, 1.0,
0.0);

```

```

begin(* getArray *)
top := 100;
numbers := worstCase;
printArray(top);
end; (* getArray *)

```

```

procedure bubbleSort(start,top: integer; var subArray: standardArray);
var

```

```

    index: integer;
    switched: boolean;
begin (* bubblesort *)
    repeat
        begin
            switched := false;
            for index := start to top-1
            do
                begin
                    if (subArray[index] > subArray[index+1])
                    then
                        begin
                            swap(subArray[index],subArray[index+1]);
                            switched := true;
                        end;
                    end;
                end;
            until switched = false;
        end;
    end;
end;

```

```

procedure findMedian(start, top: integer; var subArray: standardArray);
var

```

```

    middle : integer;
    sorted: standardArray;
begin (*findMedian *)
    middle := (start+top)div 2;
    sorted[1] := subArray[start];
    sorted[2] := subArray[top];
    sorted[3] := subArray[middle];
    bubbleSort(1,3,sorted);
    if sorted[2] = subArray[middle]
    then
        swap(subArray[start],subArray[middle])
    else
        if sorted[2] = subArray[top]
        then
            swap(subArray[start],subArray[top]);
        end;
    end;
end;

```

```

procedure sortSection(start,top:integer);
var

```

```

    swapUp: boolean;
    s,e,m: integer;
begin
    if top-start < 6
    then
        bubbleSort(start,top,numbers)
    else
        begin
            findMedian(start,top,numbers);
            swapUp := true;

            s := start;
            e := top;
            m := start;
            while e > s

```

(continued)

```

do
begin
if swapUp = true
then
begin
while (numbers[e] >= numbers[m]) and (e > m)
do
e := e - 1;
if e > m
then
begin
swap(numbers[e], numbers[m]);
m := e;
end;
swapUp := false;
end
else
begin
while (numbers[s] <= numbers[m]) and (s < m)
do
s := s + 1;
if s < m
then
begin
swap(numbers[s], numbers[m]);
m := s;
end;
swapUp := true;
end;
end;
SortSection(start, m-1);
SortSection(m+1, top);
end;
end; (* sortsection *)
begin
getArray(last);
sortSection(0, last);
printArray(last);
end.

```

tankard.lbr

BINARY
"The Literary Detective," Jim Tankard.
February, page 231. Download lu.exe to unpack this library.

```

0REM  FREQUENCY ANALYZER 1
20 REM
30 REM  BY JIM TANKARD
40 REM  3003 CHERRY LANE
50 REM  AUSTIN, TEXAS 78703
60 REM
70 DIM A$(26), B$(26): REM  THESE ARRAYS HOLD
    THE TWO FREQUENCY COUNTS TO BE COMPARED
80 LET S1% = 0
90 D$ = CHR$(4)
100 REM  GETS NAMES OF FILES
110 INPUT "WHAT FILE CONTAINS THE FIRST
    FREQUENCY COUNT?"; N$
120 INPUT "WHAT FILE CONTAINS THE SECOND
    FREQUENCY COUNT?"; M$
130 REM  OPENS FIRST FILE AND READS ITS
    FREQUENCIES INTO A$(1)
140 PRINT D$; "OPEN "; N$
150 PRINT D$; "READ "; N$
160 FOR I = 1 TO 26
170 INPUT C$: LET C% = VAL (C$)
180 LET A$(I) = C%
190 NEXT I
200 PRINT D$; "CLOSE "; N$
210 HOME
220 REM  OPENS SECOND FILE AND READS ITS
    FREQUENCIES INTO B$(1)
230 PRINT D$; "OPEN "; M$

```



```

240 PRINT D$;"READ ";M$
250 FOR I = 1 TO 26
260 INPUT C$: LET C% = VAL (C$)
270 LET B%(I) = C%
280 NEXT I
290 PRINT D$;"CLOSE ";M$
300 REM COMPUTES DIFFERENCE INDEX
310 FOR I = 1 TO 26
320 LET SI% = SI% + ABS (A%(I) - B%(I))
330 NEXT I
340 REM PRINTS RESULTS
350 HOME
360 PRINT "FIRST SAMPLE: ";N$
370 PRINT
380 PRINT "SECOND SAMPLE: ";M$
390 PRINT
400 PRINT "DIFFERENCE INDEX: ";SI%
410 END

```

```

0REM FREQUENCY ANALYZER 2
20 REM
30 REM BY JIM TANKARD
40 REM 3003 CHERRY LANE
50 REM AUSTIN, TEXAS 78703
60 REM
70 DIM A%(26,26),B%(26,26): REM THESE ARRAYS
HOLD THE TWO FREQUENCY COUNTS TO BE COMPARED
80 LET SI% = 0
90 D$ = CHR$(4)
100 REM GETS NAMES OF FILES
110 INPUT "WHAT FILE CONTAINS THE FIRST
FREQUENCY COUNT?";N$
120 INPUT "WHAT FILE CONTAINS THE SECOND
FREQUENCY COUNT?";M$
130 REM OPENS FIRST FILE AND READS ITS
FREQUENCIES INTO A%(I,J)
140 PRINT D$;"OPEN ";N$
150 PRINT D$;"READ ";N$
160 FOR I = 1 TO 26
170 FOR J = 1 TO 26
180 INPUT C$: LET C% = VAL (C$)
190 LET A%(I,J) = C%
200 NEXT
210 NEXT
220 PRINT D$;"CLOSE ";N$
230 HOME
240 REM OPENS SECOND FILE AND READS ITS
FREQUENCIES INTO B%(I,J)
250 PRINT D$;"OPEN ";M$
260 PRINT D$;"READ ";M$
270 FOR I = 1 TO 26
280 FOR J = 1 TO 26
290 INPUT C$: LET C% = VAL (C$)
300 LET B%(I,J) = C%
310 NEXT
320 NEXT
330 PRINT D$;"CLOSE ";M$
340 REM COMPUTES DIFFERENCE INDEX
350 FOR I = 1 TO 26
360 FOR J = 1 TO 26
370 LET SI% = SI% + ABS (A%(I,J) - B%(I,J))
380 NEXT
390 NEXT
400 REM PRINTS RESULTS
410 HOME
420 PRINT "FIRST SAMPLE: ";N$
430 PRINT
440 PRINT "SECOND SAMPLE: ";M$
450 PRINT
460 PRINT "DIFFERENCE INDEX: ";SI%
470 END

```

(continued)

```

10 REM TEXT GOBBLER 1
20 REM
30 REM BY JIM TANKARD
40 REM 3003 CHERRY LANE
50 REM AUSTIN, TEXAS 78703
60 REM
70 DIM Z(100): REM STORES INITIAL
FREQUENCIES OF LETTERS
80 DIM ZX(100): REM STORES FREQUENCIES
OF LETTERS AFTER NORMALIZING TO 1000 LETTERS
90 DIM Z%(100): REM STORES FREQUENCIES OF
LETTERS AS INTEGERS NORMALIZED TO 1000 LETTERS
100 DIM Z$(100): REM STORES FREQUENCIES OF
LETTERS IN STRINGS
110 R$ = CHR$ (13)
120 D$ = CHR$ (4)
130 AS$ = CHR$ (42)
140 TEXT : HOME
150 PRINT " TEXT GOBBLER 1"
160 PRINT
170 PRINT "THIS PROGRAM COUNTS THE SINGLE LETTER
FREQUENCIES IN A SAMPLE OF PROSE TEXT."
180 PRINT
190 PRINT "IT WILL THEN PRINT OUT A TABLE OF
THOSE FREQUENCIES ON YOUR PRINTER."
200 PRINT
210 PRINT "IT WILL ALSO STORE THE FREQUENCIES
IN A FILE SO THEY CAN BE COMPARED WITH THE
FREQUENCIES FROM ANOTHER SAMPLE BY MEANS OF
'FREQUENCY ANALYZER 1'."
220 PRINT
230 INPUT "WHAT FILE CONTAINS THE TEXT YOU WANT
TO ANALYZE? ";N$
240 PRINT D$;"OPEN ";N$
250 PRINT D$;"READ ";N$
260 FOR I = 1 TO 100
270 LET ST = 0
280 LET A$ = ""
290 : GOSUB 1000
300 PRINT A$
310 NEXT I
320 PRINT D$;"CLOSE ";N$
330 PRINT CHR$ (7): REM RINGS BELL WHEN
THROUGH READING TEXT
340 GOSUB 5000
350 TEXT : END
1000 REM READS TEXT 240 CHARACTERS AT A TIME
1010 GET C$: PRINT C$;
1020 LET ST = ST + 1
1030 IF C$ = R$ THEN RETURN
1040 GOSUB 2000
1050 A$ = A$ + C$
1060 IF ST = 240 THEN RETURN
1070 GOTO 1000
2000 REM COUNTS FREQUENCIES OF CHARACTERS,
SPACES AND LETTERS
2010 LET X1 = ASC (C$)
2020 LET C1 = C1 + 1: REM COUNTS TOTAL
NUMBER OF CHARACTERS
2030 IF X1 = 32 THEN LET B1 = B1 + 1: GOTO
2080: REM COUNTS TOTAL NUMBER OF SPACES
2040 IF X1 > 96 AND X1 < 123 THEN LET X1 = X1 -
32: REM ELIMINATES LOWER CASE LETTERS
2050 IF X1 > 64 AND X1 < 91 THEN LET CH =
CH + 1: REM COUNTS TOTAL NUMBER OF LETTERS
2060 LET Z(X1) = Z(X1) + 1: REM INCREMENTS
ARRAY ELEMENT SERVING AS COUNTER
2070 IF C$ = AS$ THEN GOTO 320: REM LOOKS
FOR ASTERISK AT END OF TEXT
2080 RETURN
5000 REM NORMALIZES FREQUENCIES TO SAMPLE OF
1000 AND GETS THEM READY TO STORE
5010 FOR I = 65 TO 90
5020 LET ZX(I) = (Z(I) / CH) * 1000: REM
NORMALIZES TO SAMPLE OF 1000
5030 LET Z%(I) = INT (ZX(I) + .5): REM

```



```

CHANGES FREQUENCY TO INTEGER      VALUE
5040 LET Z$(I) = STR$(Z%(I)): REM  CHANGES
      INTEGER TO STRING
5050 NEXT I
6000 REM  PRINTS MENU
6010 PRINT : PRINT
6020 PRINT "          PRESS ANY KEY TO CONTINUE.":
      GET Q$: PRINT Q$
6030 HOME
6040 PRINT " WHICH DO YOU WANT TO DO?"
6050 PRINT
6060 PRINT " 1.  PRINT THE LETTER FREQUENCY COUNT."
6070 PRINT
6080 PRINT " 2.  STORE THE LETTER FREQUENCY COUNT
      IN A DISK FILE."
6090 PRINT
6100 PRINT " 3.  RUN THIS PROGRAM AGAIN TO
      GET THE FREQUENCY COUNT FOR
      ANOTHER SAMPLE."

6110 PRINT
6120 PRINT " 4.  RUN 'FREQUENCY ANALYZER 1'."
6130 PRINT
6140 PRINT " 5.  EXIT THIS PROGRAM."
6150 PRINT
6160 PRINT "          CHOOSE 1, 2, 3, 4 OR 5."
6170 GET Q$: PRINT
6180 IF Q$ = "1" THEN GOSUB 8000
6190 IF Q$ = "2" THEN GOSUB 10000
6200 IF Q$ = "3" THEN PRINT D$;"RUN TEXT GOBBLER 1"
6210 IF Q$ = "4" THEN PRINT D$;"RUN
      FREQUENCY ANALYZER 1"
6220 IF Q$ = "5" THEN END
6230 GOTO 6030
8000 REM  PRINTS FREQUENCY RESULTS WITH PRINTER
8010 HOME
8020 PRINT "TURN ON YOUR PRINTER, THEN HIT
      ANY KEY.": GET Q$: PRINT Q$
8030 PR# 1
8040 PRINT "          FILENAME          ";N$
8050 PRINT
8060 PRINT "          NUMBER OF CHARACTERS    ";C1
8070 PRINT
8080 PRINT "          NUMBER OF SPACES          ";B1
8090 PRINT
8100 PRINT "          LETTER          ACTUAL      NORMALIZED"
8110 PRINT
8120 FOR I = 65 TO 90
8130 PRINT "          "; CHR$(I),Z(I),Z%(I)
8140 NEXT I
8150 PR# 0
8160 RETURN
10000 REM  STORES THE FREQUENCY MATRIX ON
      A DISK FILE
10010 HOME
10020 INPUT "WHAT NAME DO YOU WANT TO GIVE
      TO THE FILE?";N$
10030 PRINT D$;"OPEN ";N$
10040 PRINT D$;"WRITE ";N$
10050 FOR I = 65 TO 90: PRINT Z$(I): NEXT I
10060 PRINT D$;"CLOSE"
10070 RETURN

```

```

10 REM  TEXT GOBBLER 2
20 REM
30 REM  BY JIM TANKARD
40 REM  3003 CHERRY LANE
50 REM  AUSTIN, TEXAS 78703
60 REM
70 LET W1 = 32
80 DIM Z(26,26): REM  STORES INITIAL
      FREQUENCIES OF LETTERS
90 DIM ZX(26,26): REM  STORES FREQUENCIES
      OF LETTERS AFTER NORMALIZING TO 10000 LETTERS
100 DIM Z%(26,26): REM  STORES FREQUENCIES
      OF LETTERS AS INTEGERS NORMALIZED TO 10000 LETTERS

```

(continued)

```

110 DIM Z$(26,26): REM STORES FREQUENCIES OF
    LETTERS IN STRINGS
120 DIM ZP$(26,26): REM STORES FREQUENCIES
    OF LETTERS IN STRINGS WITH SPACES SO TABLES
    WILL BE LINED UP
130 R$ = CHR$(13)
140 D$ = CHR$(4)
150 A$ = CHR$(42)
160 TEXT : HOME
170 PRINT "          TEXT GOBBLER 2"
180 PRINT
190 PRINT "THIS PROGRAM COUNTS THE LETTER
    PAIR FREQUENCIES IN A SAMPLE OF PROSE TEXT."
200 PRINT
210 PRINT "IT WILL THEN PRINT OUT A TABLE
    OF THOSE FREQUENCIES ON YOUR PRINTER."
220 PRINT
230 PRINT "IT WILL ALSO STORE THE FREQUENCIES
    IN A FILE SO THEY CAN BE COMPARED WITH
    THE FREQUENCIES FROM ANOTHER SAMPLE BY
    MEANS OF 'FREQUENCY ANALYZER 2.'"
240 PRINT
250 INPUT "WHAT FILE CONTAINS THE TEXT YOU WANT
    TO ANALYZE? ";N$
260 PRINT D$;"OPEN ";N$
270 PRINT D$;"READ ";N$
280 FOR I = 1 TO 100
290 LET ST = 0
300 LET A$ = ""
310 : GOSUB 1000
320 : PRINT A$
330 NEXT I
340 PRINT D$;"CLOSE ";N$
350 GOSUB 5000
360 TEXT : END
1000 REM READS TEXT 240 CHARACTERS AT A TIME
1010 GET C$: PRINT C$;
1020 LET ST = ST + 1
1030 IF C$ = R$ THEN RETURN
1040 GOSUB 2000
1050 A$ = A$ + C$
1060 IF ST = 240 THEN RETURN
1070 GOTO 1000
2000 REM COUNTS FREQUENCIES OF CHARACTERS,
    SPACES AND LETTER PAIRS
2010 LET X1 = ASC (C$)
2020 LET C1 = C1 + 1: REM COUNTS TOTAL
    NUMBER OF CHARACTERS
2030 IF X1 = 32 THEN LET B1 = B1 + 1: REM
    COUNTS TOTAL NUMBER OF SPACES
2040 IF X1 > 96 AND X1 < 123 THEN LET X1 =
    X1 - 32: REM ELIMINATES LOWER CASE LETTERS
2050 IF X1 > 64 AND X1 < 91 THEN LET CH =
    CH + 1: REM COUNTS TOTAL NUMBER OF LETTERS
2060 IF C$ = A$ THEN GOTO 340: REM LOOKS
    FOR ASTERISK AT END OF TEXT
2070 IF X1 < 65 OR X1 > 90 THEN LET W1 = 32:
    GOTO 2120: REM IF CHARACTER IS NOT A
    LETTER, ASSIGNS 32 TO W1
2080 LET X1 = X1 - 64: REM CHANGES ASCII
    NUMBER OF LETTER TO 1-26 RANGE
2090 IF W1 = 32 THEN GOTO 2110: REM SKIPS
    COUNTER IF CHARACTER IS NOT A LETTER
2100 LET Z(W1,X1) = Z(W1,X1) + 1: REM INCREMENTS
    ARRAY ELEMENT SERVING AS LETTER PAIR COUNTER
2110 LET W1 = X1: REM STORES NUMBER FOR OLD
    LETTER IN W1 BEFORE GETTING NEXT CHARACTER
2120 RETURN
5000 REM NORMALIZES FREQUENCIES AND GETS THEM
    READY TO STORE

5010 FOR I = 1 TO 26
5020 FOR J = 1 TO 26
5030 LET ZX(I,J) = (Z(I,J) / CH) * 10000: REM
    NORMALIZES TO SAMPLE OF 10000
5040 LET Z%(I,J) = INT (ZX(I,J) + .5): REM

```



```

CHANGES FREQUENCY TO INTEGER VALUE
5050 LET Z$(I,J) = STR$ (Z%(I,J)): REM CHANGES
      INTEGER TO STRING
5060 REM ADDS SPACES TO STRINGS SO TABLE
      WILL BE LINED UP
5070 LET ZP$(I,J) = Z$(I,J)
5080 IF LEN (ZP$(I,J)) = 1 THEN LET ZP$(I,J) =
      " " + ZP$(I,J)
5090 IF LEN (ZP$(I,J)) = 2 THEN LET ZP$(I,J) =
      " " + ZP$(I,J)
5100 NEXT
5110 NEXT
5120 PRINT CHR$ (7): REM RINGS BELL WHEN
      THROUGH READING TEXT
7000 REM PRINTS MENU
7010 PRINT : PRINT
7020 PRINT "      PRESS ANY KEY TO CONTINUE.":
      GET Q$: PRINT Q$
7030 HOME
7040 PRINT "      WHICH DO YOU WANT TO DO?"
7050 PRINT
7060 PRINT "      1. PRINT THE LETTER FREQUENCY
          COUNT."
7070 PRINT
7080 PRINT "      2. STORE THE LETTER FREQUENCY
          COUNT IN A DISK FILE."
7090 PRINT
7100 PRINT "      3. RUN THIS PROGRAM AGAIN TO
          GET THE FREQUENCY COUNT FOR
          ANOTHER SAMPLE."
7110 PRINT
7120 PRINT "      4. RUN 'FREQUENCY ANALYZER 2'."
7130 PRINT
7140 PRINT "      5. EXIT THIS PROGRAM."
7150 PRINT
7160 PRINT "      CHOOSE 1, 2, 3, 4 OR 5."
7170 GET Q$: PRINT
7180 IF Q$ = "1" THEN GOSUB 10000
7190 IF Q$ = "2" THEN GOSUB 13000
7200 IF Q$ = "3" THEN PRINT D$;"RUN
      TEXT GOBBLER 2"
7210 IF Q$ = "4" THEN PRINT D$;"RUN
      FREQUENCY ANALYZER 2"
7220 IF Q$ = "5" THEN END
7230 GOTO 7030
10000 REM PRINTS FREQUENCY RESULTS WITH PRINTER
10010 TEXT : HOME
10020 PRINT "TURN ON YOUR PRINTER, THEN HIT
      ANY KEY.": GET Q$: PRINT Q$
10030 PR# 1
10040 PRINT CHR$ (9);"80N"
10050 PRINT CHR$ (27);"Q"
10060 HTAB 33: PRINT "FILENAME      ";N$
10070 PRINT
10080 HTAB 33: PRINT "NUMBER OF CHARACTERS ";C1
10090 PRINT
10100 HTAB 33: PRINT "NUMBER OF SPACES  ";B1
10110 PRINT
10120 HTAB 21: PRINT "FIRST LETTER
      SECOND LETTER"
10130 PRINT
10140 HTAB 33
10150 FOR I = 1 TO 26
10160 PRINT " "; CHR$ (64 + I);" ";
10170 NEXT I
10180 PRINT
10190 FOR I = 1 TO 26
10200 PRINT " "; CHR$ (64 + I);" ";
10210 FOR J = 1 TO 26
10220 PRINT ZP$(I,J);" ";
10230 NEXT
10240 NEXT
10250 PRINT
10260 PR# 0
10270 RETURN

```

(continued)

```

13000 REM STORES THE FREQUENCY MATRIX ON A
      DISK FILE
13010 HOME
13020 INPUT "WHAT NAME DO YOU WANT TO GIVE TO THE FILE?";N$
13030 PRINT D$;"OPEN ";N$
13040 PRINT D$;"WRITE ";N$
13050 FOR I = 1 TO 26
13060 FOR J = 1 TO 26
13070 PRINT Z$(I,J)
13080 NEXT
13090 NEXT
13100 PRINT D$;"CLOSE"
13110 RETURN

```

scanpoem.doc

TEXT

"Poetry Processing" by Michael Newman
 February, page 221. Also download scanpoem.exe and scanpoem.pas.

I have provided a Pascal program (Editor's note: The Microsoft Pascal source code and executable version are available from BYTEnet Listings, xxx-xxx-xxxx, as SCANPOEM.PAS and SCANPOEM.EXE. The executable version requires any MS-DOS or PC-DOS machine) that implements the syllabification algorithm and illustrates how the Poetry Processor "reads" a user's poem according to a user-specified metric scheme.

The present algorithm is not perfect, but it produces a readable, if not dictionary-perfect, syllabified word 95% of the time. To run the program, prepare two files. TEST.POE must contain the lines of poetry. You can write TEST.POE as a text file with each line of the poem on a separate line. A second text file, TEST.FRM, should have a line containing a string of dots (.) and dashes (-) indicating the accentual scheme that each line of poetry is supposed to follow. Slashes indicating the end of a foot are optional.

As an example, a Shakespearean sonnet (iambic pentameter)

will have a TEST.FRM file consisting of 14 lines of .-/.-/.-/.-/. Each line in TEST.FRM must end with an asterisk. After editing the TEST.FRM and TEST.POE files, you can run the program by entering its name, SCANPOEM. The computer will "read" the poem, printing in upper case the appropriately stressed syllables.

Note that the program is a prototype version of the algorithm. It will not handle text with capital letters, apostrophes, or punctuation, so be careful not to include them these features in TEST.POE. When using this demonstration program, you will undoubtedly find that some words are not properly syllabified. I leave it to the readers to improve upon the algorithm.

0:2

Textbox: Machine Reading Of Metric Verse
 by Paul Holzer

A computer can definitively scan a line of poetry for its stress pattern principally in one of two ways: (1) an algorithm can deduce the syllabic structure and the stressed syllables from analysis of the letters that make up the word, or (2) the computer can look-up of every word in a dictionary database that holds the syllabification and accentuation of every word. The look-up method requires a large database, and the algorithmic approach is complex and requires a deep analysis of English phonetics and spelling.

One of the features of a poetry processor is that the poet-user can specify the meter of every line of a poem. For example, the string .-/.-/.-/.-/.- represents iambic

pentameter. Dots "." indicate an unstressed syllable and dashes "-" represent a stressed one. The slash "/" indicates the end of a foot, the basic metric unit. The opening line of Shakespeare's 18th Sonnet

shall I comPARE thee TO a SUMmer's DAY?

is an example of a line of iambic pentameter with the stressed syllables are in upper case.

After writing a poem, users might request a metric scan of the poem. I will describe here a method for doing this which is not based on one of the two general solutions I mentioned in the first paragraph. Instead, the processor will break each word into its syllables and then redisplay each line, with each syllable in upper or lower case according to the position of the dots and dashes in a user-specified metric form. So, were Shakespeare trying to compose trochaic pentameter, with the metric pattern -./-./-./-./, the processor would reply with

SHALL i COMPare THEE to A sumMER'S day?

He would read this to himself, trying to put the stress on the upper case syllables. Noting the rhythmic clumsiness, he might rewrite his line as follows:

To a summer's day I shall compare thee

and the processor would respond:

TO a SUMmer's DAY i SHALL comPARE thee.

Sounds better!

The main task for the computer is to break each word into its syllables. The algorithm is based on a systematic application of what appear to be the general rules by which

English words break into syllables. Of course, there are no fixed rules, as evidenced by the fact that different dictionaries give different syllabifications for the same word.

The following is a simple version of the algorithm:

1. Break the word up into a sequence of alternating vowel and consonant groupings. Thus MICROCOMPUTER becomes M I CR O C O MP U T E R. Wherever there is a vowel or group of contiguous vowels, there will be a syllable. We need only assign the neighboring consonants to the syllable on right or to the one on the left.
2. If the first vowel-group has a consonant-group to its left, then assimilate this consonant-group to the vowel-group. This leads, in our example, to MI CR O C O MP U T E R.
3. If the final vowel-group has a consonant-group to its right, then assimilate this consonant-group to the vowel-group. We now get MI CR O C O MP U T ER.
4. For the remaining unassigned consonants, do the following:
 - a) if the consonant stands alone, attach it to the following vowel. Thus we get MI CR O CO MP U TER.
 - b) if there are two consonants, split them. We get MIC RO COM PU TER.
 - c) if there are three consonants, then
 - i) if there is a doubled consonant, then split the pair; thus APPLY -> A PPL Y -> AP PLY.

(continued)


```

13000 REM
      DISK
13010 HOME
13020 INPUT
13030 PRINT
13040 PRINT
13050 FOR :
13060 FOR :
13070 PRINT
13080 NEXT
13090 NEXT
13100 PRINT
13110 RETU

```

scanpoem.dc

TEXT
 "Poetry Pro
 February, 1

I have pro
 Pascal sou
 BYTenet Li
 SCANPOEM.E
 PC-DOS mac
 and illust
 according

The pr
 readable,
 the time.
 TEST.POE
 TEST.POE
 separate
 containin
 accentual
 follow. S

As c

will hav
 .-/.-/.-/
 asterisk.
 can run i
 computer
 appropri

Not
 It will
 or punct
 features
 you will
 syllabif
 algorith

0:2

ii) if there is no doubled consonant, but the first of the three consonants is "n", "r", or "l", then split between the second and third consonants.

iii) in all other cases, split between the first and second consonants.

Before applying this algorithm, however, we must preprocess the initial string of letters in order to take into account certain peculiarities of English orthography:

1. Final "e" is silent (with certain exceptions); treat it as a special consonant. Thus

compute -> c o m p u t e -> co m p u t e -> com p u t e.

2. Translate many two-letter sequences into special single consonants, e.g., "sh", "th", "gu", "qu", and "ck".

3. Identify common suffixes. For example, the algorithm applied to "blameless" would yield

bl a m e l e s s -> bla me less,

However, when "less" is removed as a suffix, then the "e" in "blame" would be recognized as silent, yielding blame less.

4. Identify some prefixes. For example, if "en" is recognized as a prefix, then "enact" becomes "en act", rather than "e nact".

It seems to be impossible to come up with a reasonably small set of rules and preprocessing steps to guarantee correct syllabification of all words. Two examples will illustrate some of the inherent difficulties:

1. Compound words: The algorithm will not detect the silent "e" in "snake" within the compound word "snakebite" unless the fragment "bite" is recognized as a word or treated as a suffix. Avoiding the problem would require either extensive word or prefix table look-ups.

2. Successive vowels in different syllables: In "reach", the "ea" is a single vowel sound, and the algorithm would treat it correctly. In "react", we pronounce the "e" and "a" separately, and the correct syllabification is "re act". Were the algorithm modified to isolate "re" as a prefix, it would treat "react" correctly, but turn "reach" into "re ach".

Where ambiguities can arise, the best approach is to formulate a rule that leads to the smallest number of cases requiring table look-ups for resolution.

I have provided a Pascal program (Editor's note: The Microsoft Pascal source code and executable version are available from BYTenet Listings, xxx-xxx-xxxx, as SCANPOEM.PAS and SCANPOEM.EXE. The executable version requires any MS-DOS or PC-DOS machine) that implements the syllabification algorithm and illustrates how the Poetry Processor "reads" a user's poem according to a user-specified metric scheme.

The present algorithm is not perfect, but it produces a readable, if not dictionary-perfect, syllabified word 95% of the time. To run the program, prepare two files. TEST.POE must contain the lines of poetry. You can write

TEST.POE as a text file with each line of the poem on a separate line. A second text file, TEST.FRM, should have a line containing a string of dots (.) and dashes (-) indicating the accentual scheme that each line of poetry is supposed to follow. Slashes indicating the end of a foot are optional.

As an example, a Shakespearean sonnet (iambic pentameter) will have a TEST.FRM file consisting of 14 lines of `./-./-./-./-./`. Each line in TEST.FRM must end with an asterisk. After editing the TEST.FRM and TEST.POE files, you can run the program by entering its name, SCANPOEM. The computer will "read" the poem, printing in upper case the appropriately stressed syllables.

Note that the program is a prototype version of the algorithm. It will not not handle text with capital letters, apostrophes, or punctuation, so be careful not to include them these features in TEST.POE. When using this demonstration program, you will undoubtedly find that some words are not properly syllabified. I leave it to the readers to improve upon the algorithm.

About the Author:

Paul Holzer (140 W. 16th St. Apt 3W New York, NY 10011) is a financial analyst and programmer for PaineWebber, Inc. He has a B.A. in Philosophy from Princeton University and an M.A. in Applied Mathematics from City University of New York.

scanpoem.pas

TEXT

"Poetry Processing." See scanpoem.doc.

```

program scanpoem(input,output);
{ Program written by Paul Holzer, Oct. 10, 1985 }
const maxlen = 30;
    numsub = 11;
    maxlen = 80;
    maxver = 20;
    maxsyl = 20;
    maxchn = 20;
    maxfin = maxlen+maxsyl;
    tstsize = 2;
    left = 0; right = 1;

```

```
type
charset = set of char;
```

```
var
copyright: string(36);
in_file: text;
poem: array[1..maxver] of lstring(maxlin);
form: array[1..maxver] of lstring(maxsyl);
chain: array[1..maxchn] of string(maxleng);
finline: string(maxfin);
filenom: string(12);
tword: string(maxleng);
vowels, letters, xletters: charset;
sibs, dentals: charset;
brknwrd: string(maxleng);
i, j, k, n, pos, numver, numwrds: integer;
fch, pch: char;
stress: boolean;
xgrid: array[0..12] of string(2);
sufmat: array[1..numsuf] of string(4);
inword: array[1..tstsize] of string(maxleng);
mids: charset;
rassim: array[1..29] of charset;
```

```
value
copyright := '(C) 1985 Michael Newman, Paul Holzer';
letters := ['a', 'z', 'E', 'I', 'Y'];
xletters := ['0', '9', '!', '<'];
vowels := ['a', 'e', 'i', 'o', 'u', 'y'];
```

(continued)

```

sibs := ['x','z','j','g','s','c','5','1'];
dentals := ['d','t'];
mids := ['b','d','f','g','k','l','m','n','p','s',
         't','w','0','1','3','5','6'];

xgrid[0] := 'ck'; xgrid[1] := 'ch'; xgrid[2] :=
'gh'; xgrid[3] := 'ph';
xgrid[4] := 'rh'; xgrid[5] := 'sh'; xgrid[6] :=
'th'; xgrid[7] := 'wh';
xgrid[8] := 'qu'; xgrid[9] := 'gu'; xgrid[10] :=
'si'; xgrid[11] := 'ti';
xgrid[12] := 'gn';

sufmat[1] := '**ly'; sufmat[2] := '**y';
sufmat[3] := '**er';
sufmat[4] := '*agE'; sufmat[5] := '*est';
sufmat[6] := '*ing';
sufmat[7] := 'ness'; sufmat[8] := 'less';
sufmat[9] := '*ful';
sufmat[10] := 'ment'; sufmat[11] := 'timE';

rassim[1] := ['r','l']; rassim[3] := ['r','l']; rassim[5] := ['r','l'];
rassim[6] := ['r','l','n']; rassim[10] := ['r','l']; rassim[11] := [];
rassim[12] := []; rassim[13] := []; rassim[15] := ['r','l','s','n'];
rassim[18] := ['n','m','l']; rassim[19] := ['r']; rassim[22] := ['r'];
rassim[28] := ['r','n','m','l']; rassim[29] := ['r']; rassim[24] := ['l','r'];
rassim[23] := ['r','l']; rassim[26] := ['l','r'];

procedure swapch(vars x, y: char);
var z: char;
begin
z := x; x := y; y := z
end;

function min(x, y: integer): integer;
begin
min := x;
if y < x then min := y
end;

function lowercase(ch: char): char;
{ returns lower case form of a letter }
begin
lowercase := ch;
if ord(ch) > 64 then
lowercase := chr((ord(ch) mod 32) + 96)
end;

function upcase(ch: char): char;
{ returns upper case form of a letter }
begin
upcase := ch;
if ord(ch) > 64 then
upcase := chr((ord(ch) mod 32) + 64)
end;

procedure strip(vars qword: string; var lng: integer);
{ strips leading blanks and returns length of a string of letters }
var i: integer;
tword: string(maxleng);

begin { strip }
for i := 1 to maxleng do tword[i] := qword[i];
i := 0; lng := 0;
repeat i := i + 1 until tword[i] in letters;
repeat
lng := lng + 1;
qword[lng] := tword[i];
i := i + 1;
until not (tword[i] in letters);
end { strip };

procedure sillify(consts qword: string; vars bword: string);
{ the main procedure for syllabifying a word, and described in detail
in comments below. In general, it works by initially putting the al-
ternating sequences of consonant and vowel groups of a word into the

```


matrix fiber, then attaching the consonants to the vowels, where the syllables are, according to certain rules }

```

var i,j,pos: integer;
fiber: array[1..20] of string(6);
lastcons, chng, plural, issuf: boolean;
isplur, ispast, isswap: boolean;
ch, ch2: char;
frag, suffix: string(4);
last, lastodd, lng: integer;
tword: string(maxleng);
x, m, y: char;
split: integer;

function iscons(ch: char): boolean;
{ returns true if ch is a consonant, false otherwise }
begin
  if ch in vowels then iscons := false else iscons := true
end;

function cmatch(inch: char; patch: char): boolean;
{ returns true if inch = patch or if patch = '*', false otherwise }
begin
  cmatch := false;
  if patch = '*' then cmatch := true
  else
    if inch = patch then cmatch := true
end;

function fndsuf(consts frag: string; vars suffix: string): boolean;
{ returns true if the string frag matches one of the suffixes in sufmat,
  and returns the suffix in suffix }
var i,j,k: integer;
fnd: boolean;

begin { fndsuf }
  for i := 1 to 4 do suffix[i] := ' ';
  i := 0;
  repeat i := i + 1;
    fnd := true;
    for j := 1 to 4 do
      if not cmatch(frag[j], sufmat[i,j]) then fnd := false
      else for k := 1 to 4 do suffix[k] := sufmat[i,k]
    until fnd or (i = numsuf);
  fndsuf := fnd
end { fndsuf };

function tr(m: char): integer [PURE];
begin
  if m in xletters then tr := ord(m) - 25
  else tr := ord(m) - 97
end;

procedure initfib;
{ cnverts final 'e' of tword to the "consonant" 'E', then sorts the alter-
  nating consonant/vowel groups into the rows of the 20x6 character matrix
  fiber. The last row containing a letter is returned in last }
var i,j,pos: integer;
ch: char;
lastcons: boolean;

begin { initfib }
  if tword[lng] = 'e' then
    if iscons(tword[lng-1]) then begin
      tword[lng] := 'E';
      if isplur then begin
        if tword[lng-1] in sibs then tword[lng] := 'e'
      end else
        if ispast then begin
          if tword[lng-1] in dentals then tword[lng] := 'e'
        end
    end
  end;
  for i := 1 to 20 do
    for j := 1 to 6 do

```

(continued)

```

        fiber[i,j] := ' ';
=====
the syllable separator symbol "/" is initially placed between all groups
=====
for i := 2 to 20 do fiber[i,6] := '/';
=====
blanks are ignored. Consonant groups are placed in odd-numbered
rows, vowel groups in even-numbered ones.
=====
i := 1; j := 1; pos := 1; lastcons := true;
ch := tword[pos];
while pos <= lng do
    if ch = ' ' then begin
        pos := pos + 1;
        ch := tword[pos];
        end
    else begin
        if lastcons = iscons(ch) then begin
            fiber[i,j] := ch;
            j := j + 1;
            end
        else begin
            i := i + 1; j := 1;
            fiber[i,j] := ch;
            j := j + 1;
            lastcons := not lastcons;
            end;
        pos := pos + 1;
        ch := tword[pos]
    end;
last := i;
end { initfib };

begin { sillify }
issuf := false; isplur := false; ispast := false;
isswap := false;
for i := 1 to maxleng do tword[i] := qword[i];
strip(tword, lng);
if (tword[lng] = 's') and (tword[lng-1] <> 's') then begin
    lng := lng - 1;
    isplur := true
end else
    if lng > 4 then
        if (tword[lng] = 'd') and (tword[lng - 1] = 'e') then begin
            lng := lng - 1;
            ispast := true
        end;
    if lng > 3 then
        if tword[lng] = 'e' then
            if (tword[lng-1] in ['l','r']) and not (tword[lng-2] in vowels)
            then begin
                swapch(tword[lng], tword[lng-1]);
                isswap := true
            end;
        end;
=====
tword is now scanned from right to left, and the following letter
conversions are made in the following case statement:
'gn' is converted to the xletter '<';
'gu' followed by a vowel is converted to '9';
'qu' is converted to '8';
'ch', 'gh', 'ph', 'rh', 'sh', 'th', and 'wh' are converted to
'1', '2', '3', '4', '5', '6', '7', respectively;
'ck' is converted to '0';
'i' followed by a vowel is converted to the "consonant" 'I',
but 'sI' and 'tI' are further converted to ':' and ';' ;
'y' followed by a vowel is converted to "consonant" 'Y'.
=====
for i := lng downto 2 do begin
    ch := tword[i];
    case ch of
        'n': if tword[i-1] = 'g' then begin
            tword[i-1] := '<';
            tword[i] := ' ';
            end;
        'u': if (tword[i-1] = 'g') and (tword[i+1] in vowels) then begin
            tword[i-1] := '9';

```



```

        tword[i] := ' ';
    end
else
    if tword[i-1] = 'q' then begin
        tword[i-1] := '8';
        tword[i] := ' ';
    end;
'h': begin
    chng := true;
    ch2 := tword[i-1];
    case ch2 of
        'c': tword[i-1] := '1';
        'g': tword[i-1] := '2';
        'p': tword[i-1] := '3';
        'r': tword[i-1] := '4';
        's': tword[i-1] := '5';
        't': tword[i-1] := '6';
        'w': tword[i-1] := '7';
        otherwise chng := false
    end;
    if chng then tword[i] := ' ';
end;
'k': if tword[i-1] = 'c' then begin
    tword[i-1] := '0';
    tword[i] := ' ';
end;
'i': begin
    if tword[i+1] in vowels then begin
        tword[i] := 'I';
        if tword[i-1] = 's' then begin
            tword[i-1] := ':';
            tword[i] := ' ';
        end
    else
        if tword[i-1] = 't' then begin
            tword[i-1] := ':';
            tword[i] := ' ';
        end
    end
end;
'y': if tword[i+1] in vowels then tword[i] := 'Y';
otherwise
end;
end;
pos := 0;
for i := 1 to lng do
    if tword[i] <> ' ' then begin
        pos := pos + 1;
        tword[pos] := tword[i]
    end
else lng := lng - 1;
=====
initialize syllabification by calling initfib
=====
initfib;
=====
if word has at least 5 letters, get the last four and test whether they
form a suffix by calling fndsuf. If a suffix is found, then remove it
and repeat the process, by calling initfib with the word minus its suffix.
=====
if last > 4 then begin
    for i := 1 to 4 do frag[i] := tword[lng-4+i];
    if fndsuf(frag, suffix) then begin
        for i := 1 to 4 do
            if suffix[i] <> '*' then lng := lng - 1;
        issuf := true;
        initfib
    end;
end;
fiber[last, 6] := ' ';
if odd(last) then begin
    fiber[last-1, 6] := ' ';
    lastodd := last - 2;
end
else begin

```

(continued)

```

lastodd := last - 1
end;
=====
We now inspect all consonant groups starting with the first to follow a vowel
(row 3 of fiber) and ending with the last group to precede a vowel (row
lastodd). We associate the consonant in each group with the preceding or
following vowel groups according to rules described in comments below. In
these comments, upper case letters stand for arbitrary consonants.
=====
i := 3;
while i <= lastodd do begin
  if fiber[i,2] = ' ' then { exactly 1 consonant }
    if fiber[i,1] <> 'x' then
      fiber[i,6] := ' ' { /C/ => /C }
    else fiber[i-1,6] := ' '
  else
    if fiber[i,3] = ' ' then begin { exactly 2 consonants }
      fiber[i,6] := ' '; { /CD/ => C/D }
      fiber[i-1,6] := ' ';
      fiber[i,3] := fiber[i,2];
      fiber[i,2] := '/';
    end
    else
      if fiber[i,4] = ' ' then begin { exactly 3 consonants }
        x := fiber[i,1]; m := fiber[i,2]; y := fiber[i,3];
        split := right;
        if y = m then split := left
        else
          if m in mids then
            if y in rassim[tr(m)] then split := left;
          case split of
            right: begin
              fiber[i-1,6] := ' '; { /CDE/ => CD/E }
              fiber[i,6] := ' ';
              fiber[i,4] := fiber[i,3];
              fiber[i,3] := '/';
            end;
            left: begin
              fiber[i-1,6] := ' '; { /CDE/ => C/DE }
              fiber[i,6] := ' ';
              fiber[i,4] := fiber[i,3];
              fiber[i,3] := fiber[i,2];
              fiber[i,2] := '/';
            end;
          otherwise
            end;
        end;
      else begin { 4 or 5 consonants }
        fiber[i-1,6] := ' ';
        fiber[i,6] := ' ';
        fiber[i,5] := fiber[i,4];
        fiber[i,4] := fiber[i,3];
        fiber[i,3] := '/';
      end;
    end;
    i := i + 2;
  end;
end;
=====
Put the syllabified word back together, ignoring spaces in fiber and
converting special consonants back to normal spelling.
=====
for i := 1 to maxleng do bword[i] := ' ';
pos := 0;
for i := 1 to last do
  for j := 1 to 6 do begin
    ch := fiber[i, j];
    if ch in (letters + ['/']) then begin
      pos := pos + 1;
      bword[pos] := lowercase(ch)
    end
    else
      if ch in xletters then begin
        pos := pos + 1;
        bword[pos] := xgrid[ord(ch)-48,1];
        pos := pos + 1;
        bword[pos] := xgrid[ord(ch)-48,2];
      end;
  end;
end;

```



```

    end;
=====
    Reattach the suffix, if any, as final syllable.
=====
}
if isswap then
    if bword[pos] = '/' then swapch(bword[pos-2],bword[pos-1])
    else
        swapch(bword[pos], bword[pos-1]);
    end;
if issuf then begin
    pos := pos + 1;
    bword[pos] := '/';
    for i := 1 to 4 do
        if suffix[i] <> '*' then begin
            pos := pos + 1;
            bword[pos] := suffix[i]
        end;
    end;
end;
if isplur then begin
    pos := pos + 1;
    bword[pos] := 's';
end;
if ispast then begin
    pos := pos + 1;
    bword[pos] := 'd';
end;
end { sillify };

procedure breakln(consts tline: string; var numwrds: integer);
{ breaks a line of poetry into a matrix of words, so that chain[i,j]
  is the j-th letter of the i-th word }

var i, j, pos: integer;
begin
    for i := 1 to maxchn do
        for j := 1 to maxleng do
            chain[i,j] := ' ';
        end;
    pos := 0; i := 0;
    while pos < maxlin do begin
        repeat
            pos := pos + 1
        until (pos = maxlin) or (tline[pos] in letters);
        if pos < maxlin then begin
            i := i + 1; j := 1;
            repeat
                chain[i, j] := tline[pos];
                j := j + 1; pos := pos + 1
            until (pos = maxlin) or not(tline[pos] in letters)
        end;
    end;
    numwrds := i;
end;

procedure pstress(ch: char; stress: boolean);
{ converts a letter of upper case if parameter stress is true }
begin
    if ch in letters then
        if stress then write(upcase(ch))
        else write(lowcase(ch))
    else
        write(ch)
    end;
end;

begin { scanpoem }
writeLn;
{ read in the poem }
assign(in_file, 'test.poe');
reset(in_file); i := 0;
while (not EOF(in_file)) and (i < maxver) do begin
    i := i + 1;
    readLn(in_file, poem[i])
end;
close(in_file); numver := i;
{ read in the form }
assign(in_file, 'test.frm');
reset(in_file); i := 0;

```

(continued)

```

while (not EOF(in_file)) and (i < maxver) do begin
  i := i + 1;
  readln(in_file, form[i])
end;
close(in_file);
if i < numver then numver := i;
{=====
Each line read will be processed in sequence, by calling breakin on the
line, calling sillify on each word of the line, and reconstructing the
finished line in finline. finline is then displayed with syllables in
upper case according to the pattern of dots and dashes in form.
=====}
for i := 1 to numver do begin
  for j := 1 to maxfin do finline[j] := ' ';
  breakin(poem[i], numwrds);
  pos := 0;
  for j := 1 to numwrds do begin
    sillify(chain[j], brknwrds);
    k := 0;
    repeat
      k := k + 1;
      pos := pos + 1;
      finline[pos] := brknwrds[k]
    until brknwrds[k] = ' ';
  end;
  pos := min(pos, maxfin-1);
  finline[pos] := ' '; finline[pos+1] := '*';
  j := 1; pos := 0; fch := form[i, j];
  while fch <> '*' do begin
    if fch <> '/' then begin
      if fch = '-' then stress := true
      else stress := false;
      repeat
        pos := pos + 1; pch := finline[pos];
        pstress(pch, stress)
      until pch in [' ', '/'];
    end;
    j := j + 1; fch := form[i, j]
  end;
  pos := pos + 1;
  while not (finline[pos] = '*') do begin
    write(finline[pos]); pos := pos + 1
  end;
  writeln
end;
end.

```

readsim3.me

TEXT

"A SIMPL Compiler, Part 3: Extensions." See simpl3.bix.

Guide to Part 3 of "A SIMPL Compiler." The file includes Modula-2 source code for the entire SIMPL Compiler. You will need the Monitor program from "Building a Computer in Software (October '85) and the VM2 Assembler (November '85). There are thirteen modules to this final version of the compiler.

```

CodeGen
CodeWrite
Compiler (MOD file only)
ExprParser
Init
LexAn
Node
Parser
Routines
Symbol
SymbolTable
Token
TypeChecker

```


The programs were developed using MacModula-2, but conversion to other Modula-2 systems should be straightforward. I would appreciate hearing about any conversion difficulties or bugs. You can reach me on BIX as "jba" or by U.S. mail at 1643 Cambridge St. #34, Cambridge, MA 02138. Happy Compiling!

Jonathan Amsterdam

[Editor's note: Because of the 8 character filename limit of MS-DOS, the above modules have been combined into one file. You should break them into separate files before attempting to compile them.]

natlang.lbr

BINARY

"Interpretation of Natural Language," Jordan Pollock and David L. Waltz.
February, page 189. Download lu.exe to unpack.

readnat.me

TEXT

"Interpretation of Natural Language," Jordan Pollock and David L. Waltz.
February, page 189.

This disk contains a very compact "least common denominator" activation network editor and simulator written in xlisp. The program supports a maximum of 23 nodes in XLISP.

two versions are provided; one for xlisp1.2 and another for the more modern xlisp1.4.

There are a few sample networks included. Clock is a 3 node timing "circuit", and PARSE is a network of syntax for "John ate up the street" where the minimal attachment parse is chosen. these *.net files are saved and can be loaded by the program using the "F L file" command. NOTE: THE PARSE.NET FILE TAKES APPROX. 5 MINUTES TO LOAD ON AN IBM PC.

*.plt files are created with the File Plot option they are time versus activation level graphs which can be printed on any printer. If you use the plot function without resetting activation values to 0, the plot cycles will begin with the current activation values. A '?' on the printout means that two or more nodes have the same value. The ? substitutes for an overstrike.

sa.lisp is the program for version 1.4. To load it, run xlisp, and then call (load "sa" t t) and then call (editnet)

sa12.lisp is a modified version for xlisp 1.2 using nlambda functions instead of macros, hand-expanded backquotes, and several iteration and predicate functions (from version 1.4) explicitly defined.

To run it, run xlisp 1.2, call (load "sa12") and then call (editnet)

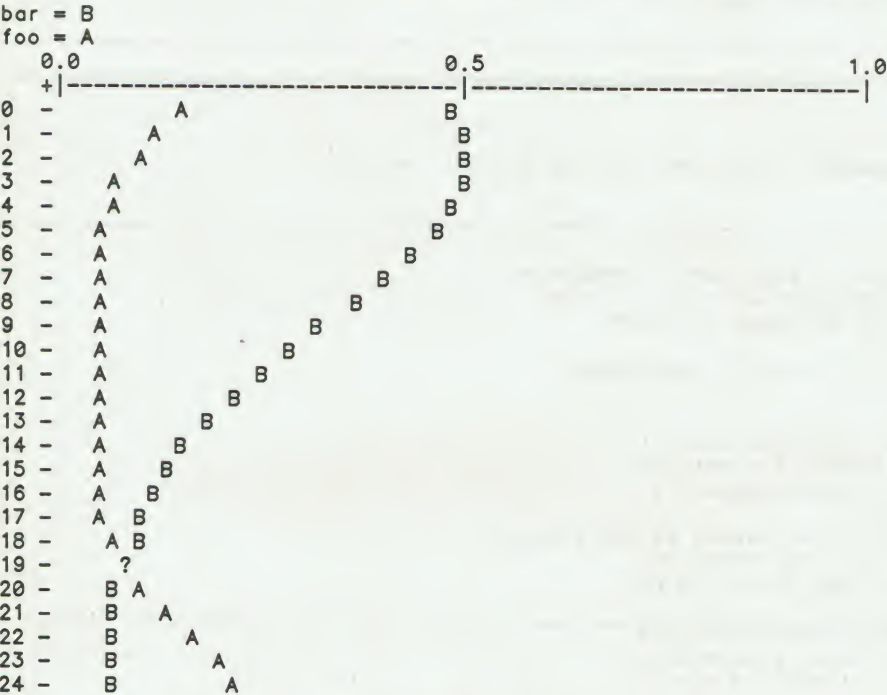
YOU MUST HAVE "device=ansi.sys" in your CONFIG.SYS FILE IN ORDER FOR THE CURSOR CONTROL FUNCTIONS TO WORK PROPERLY.

(continued)

clocknet
TEXT
"Interpretation of Natural Language," Jordan Pollock and David L. Waltz.
February, page 189.

m a foo	m l i foo zot
m s foo 15	m l i bar foo
m a bar	m l a bar zot
m s bar 49	m l a zot foo
m a zot	m l i zot bar
m l a foo bar	

clockplt
TEXT
"Interpretation of Natural Language," Jordan Pollock and David L. Waltz.
February, page 189.



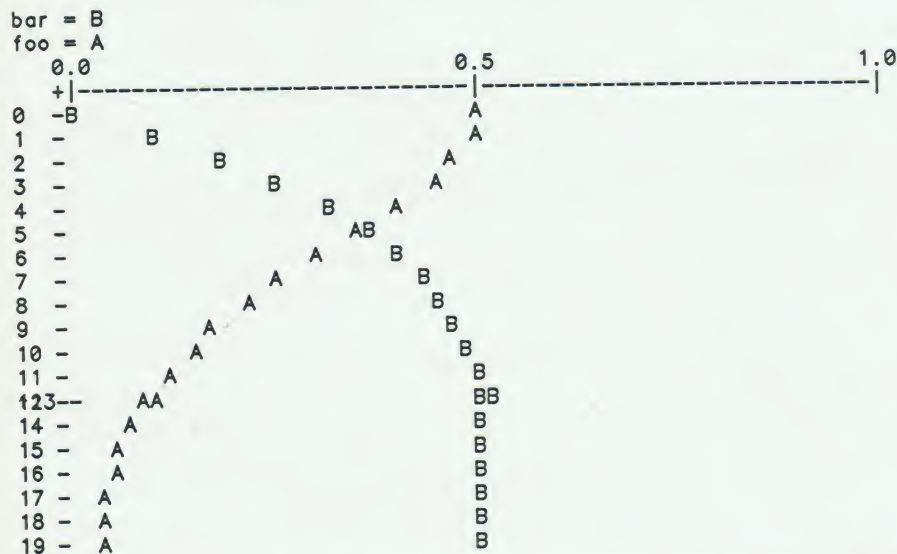
foo.net
TEXT
"Interpretation of Natural Language," Jordan Pollock and David L. Waltz.
February, page 189.

m a foo	m l a foo bar
m s foo 50	m l i bar foo
m a bar	

foo.plt

TEXT

"Interpretation of Natural Language," Jordan Pollock and David L. Waltz.
February, page 189.



parse.net

TEXT

"Interpretation of Natural Language," Jordan Pollock and David L. Waltz.
February, page 189.

```
m a john
m s john 25
m a np1
m a ate
m a v
m a up
m a adv
m a prep
m a the
m a street
m a np2
m a vp1
m a pp
m a vp2
m a s2
m a s1
m l a john np1
m l a john ate
m l a np1 john
m l a np1 s2
m l a np1 s1
m l a ate v
m l a ate up
m l a v ate
m l a v vp1
m l a v vp2
m l a up adv
m l a up prep
m l a up the
m l a adv up
m l i adv prep
```

```
m l a adv vp1
m l a prep up
m l i prep adv
m l a prep pp
m l a the street
m l a the np2
m l a street np2
m l a np2 the
m l a np2 street
m l a np2 vp1
m l a np2 pp
m l a vp1 v
m l a vp1 adv
m l a vp1 np2
m l i vp1 pp
m l i vp1 vp2
m l a vp1 s1
m l a pp prep
m l a pp np2
m l i pp vp1
m l a pp vp2
m l a vp2 v
m l i vp2 vp1
m l a vp2 pp
m l a vp2 s2
m l a s2 np1
m l a s2 vp2
m l i s2 s1
m l a s1 np1
m l a s1 vp1
m l i s1 s2
```

(continued)

 parse.plt

TEXT

 "Interpretation of Natural Language," Jordan Pollock and David L. Waltz.
 February, page 189.

```

adv = F
prep = G
vp1 = K
vp2 = M
s1 = O
s2 = N
0.0
+|-----0.5-----1.0
0 -?
1 -?
2 -?
3 -?
4 -??
5 -??
6 -? ?
7 -? ?
8 -?? ?
9 - ?MK ?
10 - G ? K ?
11 - G MF K?
12 - G M F N?
13 - G M N ? K
14 - G M N N OF O K F
15 - G M N N N O F K
16 - G M N N N O F K
17 - G M N N N O F K
18 - G M N N N O F K
19 - G M N N N O F K
20 - GM N N N O F K
21 - GM N N N O F K
22 - GM N N N O F K
23 - GM N N N O F K
24 - ? N N N O F K
      OFK

```

sa.lsp

TEXT

 "Interpretation of Natural Language," Jordan Pollock and David L. Waltz.
 February, page 189.

```

(expand 45) ; make some space
; *****
; * editnet is the main routine
; *****

```

```

(defun editnet ()
  (prog (net node lt)
    (setq net (= Net new))
    (CLS))
  loop (gc) ;force gc to avoid xlist bug
  (MENU (Quit File Modify Execute Showlinks)
    (q (return))
    (f (MENU (Load Save Clear Plot)
      (l (PUSH $FILES (openi (FNAME ".net"))))
      (s (= net SAVE))
      (c (= net CLEAR))
      (p (= net PLOT (= net SUBNET)
        (PROMPT "how many cycles")))))
    (m (MENU (Add-node Set-value Link Del-node Unlink)
      (a (setq node (PROMPT "name"))
        (= net ADD (= Node new node)))
      (s (setq node (= net FIND "node"))
        (= node ERASE)
        (= node SETVALUE (PROMPT "value"))
        (= node DRAW))
      (l (setq lt (or (MENU
        (A--> I--o S<--> Xo-o Co-->))

```



```

                (a 1)(i 2)(s 5)(x 10)(c 9))
                0))
        (=> (=> net FIND "from") LINKTO
            (=> net FIND "to") lt))
        (d (=> net DEL (=> net FIND "name")))
        (u (=> (=> net FIND "from") UNLINK
            (=> net FIND "to"))))
    (e (MENU (Reset Cycle)
        (r (=> net RESET))
        (c (=> net RUN
            (PROMPT "how many cycles")
            t nil nil))))
        (s (=> net SHOWLINKS (=> net FIND "node"))))
    (go loop)))

(defmacro MENU (items &rest actions &aux command expr)
  (setq command (FIRST-CHAR (PROMPT items)))
  (dolist (a actions expr)
    (and (= (FIRST-CHAR (car a)) command)
         (setq expr (cons 'progn (cdr a))))))

(defun FIRST-CHAR (x)
  ; first caseless char
  (bit-and 31 (ascii (symbol-name x))))

(defun PROMPT (str)
  ; read an atom
  (cond ($FILES
    (or (read (car $FILES))
        (progn (close (car $FILES))
                (setq $FILES (cdr $FILES))
                (PROMPT str))))
    (t (GOTO 1 1)
        (ERASETOEOL)
        (princ str)
        (princ "?")
        (read))))

; *****
; * support functions
; *****

(defun PLABEL (a b c char fp) ; used by PLOT
  (princ a fp)
  (dotimes (j (- (/ $PW 2) 3)) (princ char fp))
  (princ b fp)
  (dotimes (j (- (/ $PW 2) 3)) (princ char fp))
  (princ c fp)
  (terpri fp))

(defun PRINTL (l fp)
  ; print list w/o paren
  (dolist (x l) (princ x fp) (princ " " fp))
  (terpri fp))

(defun FNAME (typ)
  ; make a filename str
  (strcat (symbol-name (PROMPT "file")) typ))

(defmacro PUSH (stack item)
  ; standard macro
  '(setq ,stack (cons ,item ,stack)))

(defun PSCALE (val)
  ; (0,100)-->(0,$PW-1)
  (/ (* $PW val) $RES))

; *****
; * Global variables
; *****

(setq $RES 100) ; resolution of arith.
(setq $PW 60) ; printing width
(setq $ALV 20) ; activation link value
(setq $ILV -45) ; inhibition link value
(setq $FILES nil) ; stack of input files

```

(continued)

```

; *****
; * primitives for handling screen updates
; * below is for ANSI standard; need to personalize
; *****

(defun GOTO (lin col)                ; move the cursor
  (princ "\e[")
  (princ lin)
  (write-char 59)
  (princ col)
  (princ 'H))

(defun CLS ()                        ; clear the screen
  (princ "\e[2J"))

(defun ERASETOEOL ()                 ; ERASE to end of line
  (princ "\e[0K"))

; *****
; * macros for nonquoted object handling
; *****

(defmacro defclass                   ; define a new class
  (newclass superclass &rest ivars)
  '(progn (setq ,newclass (Class 'new))
    (,newclass 'isnew ,superclass)
    (,newclass 'ivars ',@ivars)
    '(,superclass ,newclass)))

(defmacro defmethod                   ; define a new method
  (class selector args &rest body)
  '(progn (,class 'answer ',selector ',args ',body)
    '(,class ,selector)))

(defmacro => (class selector &rest args) ;SEND message
  '(,class ',selector ,@args))

; *****
; * The Net object
; *****

(defclass Net Object
  (nodes
   line))                ; just a list of nodes
                        ; and the next display line

(defmethod Net isnew ()
  (setq line 2)
  self)

; *****
; * query methods
; *****

(defmethod Net SUBNET (&aux l n)    ; get a subnetwork
  (setq n (PROMPT "how many nodes"))
  (dotimes (i n (reverse l))
    (PUSH l (=> self FIND "node"))))

(defmethod Net FIND (str)            ; get a node by name
  (prog (out name)
    (setq name (PROMPT str))
    look (dolist (n nodes)
      (and (eq (=> n NAME?) name)
        (setq out n)))
    (and out (return out))
    (setq name (PROMPT "the name of a node"))
    (go look)))

; *****
; * method which modify networks
; *****

(defmethod Net ADD (node)             ; add a node
  (setq nodes (nconc nodes (cons node nil)))
  (=> node RENUM line)
  (setq line (1+ line)))

```



```

(=> node DRAW))

(defmethod Net DEL (node) ; delete a node
  (dolist (n nodes) (=> n UNLINK node))
  (setq nodes (delete node nodes))
  (=> self RENUMBER))

(defmethod Net CLEAR () ; empty network
  (dolist (n nodes) (=> n CLEARLINKS))
  (setq line 2)
  (setq nodes nil)
  (=> self REDRAW))

(defmethod Net RESET () ; reset all nodes
  (dolist (n nodes) (=> n RESET))
  (=> self REDRAW))

(defmethod Net RENUMBER () ; compact display
  (setq line 2)
  (dolist (n nodes)
    (=> n RENUM line)
    (setq line (1+ line)))
  (=> self REDRAW))

; *****
; * methods for displaying, plotting, and saving nets
; *****

(defmethod Net REDRAW () ; REDRAW the screen
  (CLS)
  (dolist (n nodes) (=> n DRAW)))

(defmethod Net SHOWLINKS (node) ; graph 1 node
  (CLS)
  (dolist (n nodes) (=> n SHOWTO node)))

; RUN iterates n cycles and animates and/or plots
(defmethod Net RUN (n aflag pflag fp &aux pline)
  (=> self REDRAW)
  (and pflag (dotimes (i $PW) (PUSH pline " ")))
  (dotimes (i n)
    (cond
      (pflag
       (princ (substr (strcat (itoa i) " ") 1 3)
              fp)
       (princ "-" fp)
       (mapl #'(lambda (x) (rplaca x " ")) pline)
       (dolist (n nlist) (=> n PLOT pline))
       (dolist (ch pline) (princ ch fp))
       (terpri fp)))
      (t
       (dolist (n nodes) (=> n SEND))
       (dolist (n nodes) (=> n UPDATE aflag))
       (GOTO 1 1)
       (princ i)
       (terpri)))
    )
  )

; SAVE creates a file with commands to recreate net
(defmethod Net SAVE (&aux fp)
  (setq fp (openo (FNAME ".net")))
  (dolist (n nodes) (=> n DUMP fp))
  (dolist (n1 nodes)
    (dolist (n2 nodes)
      (=> n1 DUMPLINK n2 fp)))
  (close fp))

; PLOT makes ascii printer timeline files
(defmethod Net PLOT (nlist cycles &aux fp)
  (setq fp (openo (FNAME ".plt")))
  (dolist (n nlist) (=> n LABEL fp))
  (PLABEL " 0.0" "0.5" "1.0" " " fp)
  (PLABEL " +|- " "-|- " "-| " "- " fp)
  (=> self RUN cycles nil t fp)
  (close fp)
  (=> self REDRAW))

```

(continued)

```

; *****
; * the Node object
; *****

(defclass Node Object
  (name          ; printing name
   value         ; activation value
   initval      ; initial value
   contrib      ; temp for collection
   aline       ; animation line
   ilinks      ; inhibition links
   alinks))    ; activation links

(defmethod Node isnew (nm &optional val line)
  (setq name nm)
  (setq value (setq initval (or val 0)))
  (setq contrib 0)
  (setq aline (or line 2))
  self)

; *****
; * these methods do the actual arithmetic
; * for spreading activation and lateral inhibition
; *****

(defmethod Node SEND (&aux c) ; Spread activation
  (cond ((not (zerop value))
        (setq c (* value $ALV))
        (dolist (l alinks)
          (=> l RECEIVE c))
        (setq c (* value $ILV))
        (dolist (l ilinks)
          (=> l RECEIVE c)))))

(defmethod Node RECEIVE (val) ; collect activation
  (setq contrib (+ contrib val)))

; update value
(defmethod Node UPDATE(flag &aux newval)
  (setq contrib (min $RES (max (- $RES)
                               (/ contrib $RES))))
  (setq newval
    (cond ((minusp contrib)
          (+ value (/ (* contrib value) $RES)))
          (t (+ value
                (/ (* contrib
                    (- $RES value)
                    $RES)))))
    (cond ((and flag (not (= value newval))) ; animate
          (=> self ERASE)
          (setq value newval)
          (=> self DRAW))
          (t (setq value newval)))
    (setq contrib 0))

; *****
; * these methods are for "graphic" display
; *****

(defmethod Node DRAW () ; "draw" node
  (GOTO aline (1+ (PSCALE value)))
  (princ name))

(defmethod Node ERASE () ; erase node
  (GOTO aline (1+ (PSCALE value)))
  (ERASETOEOL))

(defmethod Node SHOWTO (node &aux sum) ; grphic links
  (=> self DRAW)
  (princ " ")
  (setq sum (=> self BILINKS? node))
  (princ (nth sum '(" " --> --o --*
                  <-- <-> <-o <-*
                  o-- o-> o-o o-*
                  *-- *-> *-o *-*))
    (and (eq self node) (princ "SHOWING")))
  (terpri))

```



```

(defmethod Node PLOT (ol)          ; put a char in plot
  (rplaca (nthcdr (PSCALE value) ol)
    (or (and (= " " (nth (PSCALE value) ol))
      (chr (+ 63 aline)))
      "?"))))

(defmethod Node LABEL (fp)          ; make legend for plot
  (PRINTL '(,name = ,(chr (+ 63 aline))) fp))

; *****
; * These are "predicates" used for querying a node
; *****

(defmethod Node LINKS? (node)       ; what links to node?
  (+ (or (and (member node alinks) 1) 0)
    (or (and (member node ilinks) 2) 0)))

(defmethod Node BILINKS? (node)     ; bidirectional links?
  (+ (=> self LINKS? node)
    (* 4 (=> node LINKS? self))))

(defmethod Node NAME? ()            ; whats your name?
  name)

; *****
; * the next methods are used in saving a file
; *****

(defmethod Node DUMP (fp)            ; commands to add node
  (PRINTL '(m a ,name) fp)
  (or (= 0 value)
    (PRINTL '(m s ,name ,value) fp)))

; *****
; * commands to link
; *****
(defmethod Node DUMPLINK (node fp &aux ltype)
  (setq ltype (=> self LINKS? node))
  (or (zerop (bit-and 1 ltype))
    (PRINTL '(m l a ,name ,(=> node NAME?)) fp))
  (or (zerop (bit-and 2 ltype))
    (PRINTL '(m l i ,name ,(=> node NAME?)) fp)))

; *****
; * these methods modify nodes
; *****

(defmethod Node UNLINK (node)        ; remove 1way links
  (setq alinks (delete node alinks))
  (setq ilinks (delete node ilinks)))

(defmethod Node RESET ()             ; reset to initial val
  (setq value initval))

(defmethod Node SETVALUE (val)       ; change the value
  (setq value (setq initval val)))

(defmethod Node LINKTO (node type); create a link
  (or (zerop (bit-and 1 type))
    (setq alinks (cons node alinks)))
  (or (zerop (bit-and 2 type))
    (setq ilinks (cons node ilinks)))
  (or (zerop (bit-and 12 type))
    (=> node LINKTO self (/ type 4))))

(defmethod Node RENUM (line)         ; change display line
  (setq aline line))

(defmethod Node CLEARLINKS ()         ; fix circular
  (setq alinks (setq ilinks nil))) ; lists for gc

```

(continued)

sa12.lsp

TEXT

"Interpretation of Natural Language," Jordan Pollock and David L. Waltz.
February, page 189.

```

(expand 45)                                ; make some space
; *****
; * editnet is the main routine
; *****

(defun editnet (&aux net node lt qflag)
  (setq net (=> Net new))
  (CLS)
  (while (null qflag)
    (gc)
    (MENU (Quit File Modify Execute Showlinks)
      (q (setq qflag t))
      (f (MENU (Load Save Clear Plot)
        (l (PUSH $FILES (openi (FNAME ".net"))))
        (s (=> net SAVE))
        (c (=> net CLEAR))
        (p (=> net PLOT (=> net SUBNET)
          (PROMPT "how many cycles")))))
      (m (MENU (Add-node Set-value Link Del-node Unlink)
        (a (setq node (PROMPT "name"))
          (=> net ADD (=> Node new node 0)))
        (s (setq node (=> net FIND "node"))
          (=> node ERASE)
          (=> node SETVALUE (PROMPT "value"))
          (=> node DRAW))
        (l (setq lt (or (MENU
          (A--> I--o S<--> Xo-o Co->)
          (a 1)(i 2)(s 5)(x 10)(c 9))
          0))
          (=> (=> net FIND "from") LINKTO
            (=> net FIND "to") lt))
        (d (=> net DEL (=> net FIND "name")))
        (u (=> (=> net FIND "from") UNLINK
          (=> net FIND "to")))))
      (e (MENU (Reset Cycle)
        (r (=> net RESET))
        (c (=> net RUN
          (PROMPT "how many cycles")
          t nil nil))))
      (s (=> net SHOWLINKS (=> net FIND "node"))))
    nil))

(defun MENU (items &rest actions &aux command expr)
  (setq command (FIRST-CHAR (PROMPT items)))
  (dolist (a actions)
    (and (= (FIRST-CHAR (car a)) command)
      (setq expr (cons 'progn (cdr a)))))
  (eval expr))

(defun FIRST-CHAR (x)
  ; first caseless char
  (bit-and 31 (ascii (symbol-name x))))

(defun PROMPT (str)
  ; read an atom
  (cond ($FILES
    (or (read (car $FILES))
      (progn (close (car $FILES))
        (setq $FILES (cdr $FILES))
        (PROMPT str))))
    (t (GOTO 1 1)
      (ERASETOEOL)
      (princ str)
      (princ "?")
      (read))))

```



```

; *****
; * support functions
; *****

(defun PLABEL (a b c char fp) ; used by PLOT
  (princ a fp)
  (dotimes (j (- (/ $PW 2) 3)) (princ char fp))
  (princ b fp)
  (dotimes (j (- (/ $PW 2) 3)) (princ char fp))
  (princ c fp)
  (terpri fp))

(defun PRINTL (l fp)           ; print list w/o paren
  (dolist (x l) (princ x fp) (princ " " fp))
  (terpri fp))

(defun FNAME (typ)             ; make a filename str
  (strcat (symbol-name (PROMPT "file")) typ))

(ndefun PUSH (stack item)      ; standard macro
  (eval (list 'setq stack (list 'cons item stack))))

(defun PSCALE (val)            ; (0,100)-->(0,$PW-1)
  (/ (* $PW val) $RES))

; *****
; * Global variables
; *****

(setq $RES 100)                ; resolution of arith.
(setq $PW 60)                  ; printing width
(setq $ALV 20)                 ; activation link value
(setq $ILV -45)                ; inhibition link value
(setq $FILES nil)              ; stack of input files

; *****
; * primitives for handling screen updates
; * below is for ANSI Standard; need to personalize
; *****

(defun GOTO (lin col)          ; move the cursor
  (princ "\e[")
  (princ lin)
  (write-char 59)
  (princ col)
  (princ 'H))

(defun CLS ()                  ; clear the screen
  (princ "\e[2J"))

(defun ERASETOEOL ()           ; ERASE to end of line
  (princ "\e[0K"))

; *****
; * macros for nonquoted object handling
; *****

(ndefun defclass               ; define a new class
  (newclass superclass ivars)
  (set newclass (Class 'new))
  (eval (list newclass 'isnew superclass))
  (eval (list newclass 'ivars
    (list 'quote ivars))))

(ndefun defmethod              ; define a new method
  (class selector args &rest body)
  (eval (list class 'answer (list 'quote selector)
    (list 'quote args)
    (list 'quote body)))))

(ndefun => (class selector &rest args) ;SEND message
  (eval (cons class (cons (list 'quote selector)
    args))))

```

(continued)

```

; *****
; * The Net object
; *****

(defclass Net Object
  (nodes          ; just a list of nodes
   line)         ; and the next display line

  (defmethod Net isnew ()
    (setq line 2)
    self)

; *****
; * query methods
; *****

(defmethod Net SUBNET (&aux l n) ; get a subnetwork
  (setq n (PROMPT "how many nodes"))
  (dotimes (i n (reverse l))
    (PUSH 1 (=> self FIND "node"))))

(defmethod Net FIND (str &aux out name) ; get a node
  (while (null out)
    (setq name (PROMPT str))
    (dolist (n nodes)
      (and (eq (=> n NAME?) name)
        (setq out n)))
    (setq str "the name of a node")
    out))

; *****
; * method which modify networks
; *****

(defmethod Net ADD (node) ; add a node
  (setq nodes (nconc nodes (cons node nil)))
  (=> node RENUM line)
  (setq line (1+ line))
  (=> node DRAW))

(defmethod Net DEL (node) ; delete a node
  (dolist (n nodes) (=> n UNLINK node))
  (setq nodes (delete node nodes))
  (=> self RENUMBER))

(defmethod Net CLEAR () ; empty network
  (dolist (n nodes) (=> n CLEARLINKS))
  (setq line 2)
  (setq nodes nil)
  (=> self REDRAW))

(defmethod Net RESET () ; reset all nodes
  (dolist (n nodes) (=> n RESET))
  (=> self REDRAW))

(defmethod Net RENUMBER () ; compact display
  (setq line 2)
  (dolist (n nodes)
    (=> n RENUM line)
    (setq line (1+ line)))
  (=> self REDRAW))

; *****
; * methods for displaying, plotting, and saving nets
; *****

(defmethod Net REDRAW () ; REDRAW the screen
  (CLS)
  (dolist (n nodes) (=> n DRAW)))

(defmethod Net SHOWLINKS (node) ; graph 1 node
  (CLS)
  (dolist (n nodes) (=> n SHOWTO node)))

; RUN iterates n cycles and animates and/or plots
(defmethod Net RUN (n aflag pflag fp &aux pline)
  (=> self REDRAW))

```



```

(and pflag (dotimes (i $PW) (PUSH pline " ")))
(dotimes (i n)
  (cond
    (pflag
      (princ (substr (strcat (itoa i) " ") 1 3)
        fp)
      (princ "-" fp)
      (maplist '(lambda (x) (rplaca x " "))
        pline)
      (dolist (n nlist) (= n PLOT pline))
      (dolist (ch pline) (princ ch fp))
      (terpri fp)))
    (dolist (n nodes) (= n SEND))
    (dolist (n nodes) (= n UPDATE aflag))
    (GOTO 1 1)
    (princ i)
    (terpri)))

; SAVE creates a file with commands to recreate net
(defmethod Net SAVE (&aux fp)
  (setq fp (openo (FNAME ".net")))
  (dolist (n nodes) (= n DUMP fp))
  (dolist (n1 nodes)
    (dolist (n2 nodes)
      (= n1 DUMPLINK n2 fp)))
  (close fp))

; PLOT makes ascii printer timeline files
(defmethod Net PLOT (nlist cycles &aux fp)
  (setq fp (openo (FNAME ".plt")))
  (dolist (n nlist) (= n LABEL fp))
  (PLABEL " 0.0" "0.5" "1.0" " " fp)
  (PLABEL " +|- " "-|- " "-| " "- " fp)
  (= self RUN cycles nil t fp)
  (close fp)
  (= self REDRAW))

; *****
; * the Node object
; *****

(defclass Node Object
  (name ; printing name
    value ; activation value
    initval ; initial value
    contrib ; temp for collection
    aline ; animation line
    ilinks ; inhibition links
    alinks)) ; activation links

(defmethod Node isnew (nm val)
  (setq name nm)
  (setq value (setq initval val))
  (setq contrib 0)
  (setq aline 2)
  self)

; *****
; * these methods do the actual arithmetic
; * for spreading activation and lateral inhibition
; *****

(defmethod Node SEND (&aux c) ; Spread activation
  (cond ((not (zerop value))
    (setq c (* value $ALV))
    (dolist (l alinks)
      (= l RECEIVE c))
    (setq c (* value $ILV))
    (dolist (l ilinks)
      (= l RECEIVE c)))))

(defmethod Node RECEIVE (val) ; collect activation
  (setq contrib (+ contrib val)))

; update value

```

(continued)

```

(defmethod Node UPDATE (flag &aux newval)
  (setq contrib (min $RES (max (minus $RES)
                                (/ contrib $RES))))
  (setq newval
    (cond ((minusp contrib)
           (+ value (/ (* contrib value) $RES)))
          (t (+ value
                 (/ (* contrib
                     (- $RES value)
                     $RES)))))
    (cond ((and flag (not (= value newval))) ; animate
           (=> self ERASE)
           (setq value newval)
           (=> self DRAW))
          (t (setq value newval))))
  (setq contrib 0))

; *****
; * these methods are for "graphic" display
; *****

(defmethod Node DRAW () ; "draw" node
  (GOTO aline (1+ (PSCALE value)))
  (princ name))

(defmethod Node ERASE () ; erase node
  (GOTO aline (1+ (PSCALE value)))
  (ERASETOEOL))

(defmethod Node SHOWTO (node &aux sum) ; grphic links
  (=> self DRAW)
  (princ " ")
  (setq sum (=> self BILINKS? node))
  (princ (nth sum '(" " --> --o --*
                  <-- <-> <-o <-*
                  o-- o-> o-o o-*
                  *-- *-> *-o *-*))
    (and (eq self node) (princ "SHOWING")))
  (terpri))

(defmethod Node PLOT (ol) ; put a char in plot
  (rplaca (nthcdr (PSCALE value) ol)
    (or (and (= " " (nth (PSCALE value) ol))
              (chr (+ 63 aline)))
        "?"))))

(defmethod Node LABEL (fp) ; make legend for plot
  (PRINTL (list name '= (chr (+ 63 aline))) fp))

; *****
; * These are "predicates" used for querying a node
; *****

(defmethod Node LINKS? (node) ; what links to node?
  (+ (or (and (member node alinks) 1) 0)
     (or (and (member node ilinks) 2) 0)))

(defmethod Node BILINKS? (node) ; bidirectional links?
  (+ (=> self LINKS? node)
     (* 4 (=> node LINKS? self))))

(defmethod Node NAME? () ; whats your name?
  name)

; *****
; * the next methods are used in saving a file
; *****

(defmethod Node DUMP (fp) ; commands to add node
  (PRINTL (list 'm 'a name) fp)
  (or (= 0 value)
    (PRINTL (list 'm 's name value) fp)))

; *****
; * commands to link
; *****
(defmethod Node DUMPLINK (node fp &aux ltype)
  (setq ltype (=> self LINKS? node))
  (or (zerop (bit-and 1 ltype))
    (PRINTL (list 'l ltype) fp)))

```



```

(PRINTL (list 'm 'l 'a name (=> node NAME?))
fp))
(or (zerop (bit-and 2 ltype))
(PRINTL (list 'm 'l 'i name (=> node NAME?))
fp)))

; *****
; * these methods modify nodes
; *****

(defmethod Node UNLINK (node) ; remove 1way links
  (setq alinks (delete node alinks))
  (setq ilinks (delete node ilinks)))

(defmethod Node RESET () ; reset to initial val
  (setq value initval))

(defmethod Node SETVALUE (val) ; change the value
  (setq value (setq initval val)))

(defmethod Node LINKTO (node type); create a link
  (or (zerop (bit-and 1 type))
      (setq alinks (cons node alinks)))
  (or (zerop (bit-and 2 type))
      (setq ilinks (cons node ilinks)))
  (or (zerop (bit-and 12 type))
      (=> node LINKTO self (/ type 4))))

(defmethod Node RENUM (line) ; change display line
  (setq aline line))

(defmethod Node CLEARLINKS () ; fix circular
  (setq alinks (setq ilinks nil))) ; lists for gc

; *****
; * XLISP 1.4 quasi-compatible functions
; *****

(ndefun dolist (iexp &rest exprs)
  (mapcar (cons 'lambda (cons (list (car iexp))
                                exprs))
          (eval (cadr iexp)))
  nil)

(ndefun dotimes (iexp &rest exprs)
  (eval (list 'progn
              (list 'setq (car iexp) -1)
              (cons 'repeat
                    (cons (cadr iexp)
                          (cons (list 'setq
                                      (car iexp)
                                      (list '1+ (car iexp)))
                                exprs))))))

(defun zerop (x) (= x 0))
(defun minusp (x) (< x 0))

```

stramdisk.c

TEXT
 "Programming Tools and the Atari 520st", Bruce Webster.
 February, page 331. Also download ST.doc

ALERT /* ALERT /* ALERT

DOWNLOAD! ST.doc for the revised ST upgrade procedure
 DOWNLOAD! ST.doc for the revised ST upgrade procedure
 DOWNLOAD! ST.doc for the revised ST upgrade procedure

(continued)

----- /*
 RAM DISK ACCESSORY version 1.1 - sept 10, 1985,
 Gert Slavenburg, Mountain View, CA.

For Atari 520ST with RAM extension to 1 MByte.
 Link this program with accstart,%1,osbind,vdibind,aesbind

This is the first, experimental version of the RAMdisk. It should be installed as a DESK ACCESSORY on the Bootdisk. (that is it should be called either DESK1, DESK2, or DESK3.ACC). Before re-booting, and thus activating it, the menu entry "install disk" should be used to install drive "D" with icon-label "RAMDISK". Then save the desktop to make the newly installed drive permanent. Now re-boot end enjoy. (write protect if you're paranoid - I am)

This version (1.1) avoids any directory knowledge by just copying the whole disk that it is booted on into the RAMdisk. This makes booting seem to take forever. Throw away any files on the RAMdisk that you don't want by just deleting them !

You may get annoyed by my trick to let the RAMdisk give a beep every 30 seconds. This was a debugging tool for me to see if it's still alive, and I get worried now if I don't hear it anymore... To remove it, just replace the "beep" statement in procedure "sleep".

known bugs :

- 1) GEMDOS refuses to do full diskette copy to/from the RAMdisk. Don't know why it does that, since the BPB's are identical.

Lots of fun - Gert

*/

```
#include "portab.h"
#include "obdefs.h"
#include "define.h"
#include "gemdefs.h"
#include "osbind.h"
```

```
struct bpb
{
  .More..
  { WORD recsiz, /* see BIOS:rwabs.c for more info */
    clisiz,
    cliszb,
    rdlen,
    fsiz,
    fatrec,
    datrec,
    numcl,
    bflags;
  };
};
```

/* stupid AES binding arrays - what a drag */

```
int contrl[12]; /* better find out what's REALLY NEEDED */
int intin[128];
int ptsin[128];
int intout[128];
int ptsout[128];
```

/* the variables below are really serious */

```
typedef LONG (*PFL)(); /* define "pointer to function returning a long" */
typedef WORD (*PFW)(); /* define "pointer to function returning a word" */
```

```
PFL getbpb; /* pointer to the systems original getbpb function */
PFW mediach; /* pointer to the systems original mediach */
PFL rwabs; /* pointer to the systems original rwabs */
```

```
struct bpb rdiskbpb = { 512, 2, 1024, 7, 5, 6, 18, 351, 0 };
/* same as Single Sided microdiskette */
```

```
int data[184320]; /* 720 sectors of 512 bytes */
/* same as Single Sided microdiskette */
```



```

LONG RDgetbpb(dev)
WORD dev;
{
    if (dev != 3)
        return( (*getbpb)(dev) ); /* pass all non-RAMdisk to old handler */
    else
        return( &rdiskbpb ); /* return our bpb */
}

WORD RDmediach(dev)
WORD dev;
{
    if (dev != 3)
        return(( *mediach)(dev) ); /* pass all non-RAMdisk to old handler */
    else
        return( 0 ); /* RAMDISK media never changes */
}

LONG RDrwabs(rw,buf,count,recno,dev)
WORD rw;
int *buf;
WORD count, recno, dev;
{
    int i, *p;

    if (dev != 3)
        return( (*rwabs)(rw,buf,count,recno,dev) ); /* pass it on */
    else
    {
        if (rw > 1) rw -= 2; /* we never change media anyway */
        while (count > 0)
        {
            p = &data[(((long) recno) * 256L)]; /* both casts necessary - C068 bug */
            if (rw==0) /* read */
                for (i=0; i<256; i++) *buf++ = *p++;
            else /* write */
                for (i=0; i<256; i++) *p++ = *buf++;
            count--; recno++;
        }
        return(0L);
    }
}

install() /* take over DISKIO vectors, MUST RUN AS SUPERVISOR */
{
    long *bpbvect = 0x472;
    long *rwvect = 0x476;
    long *mcvect = 0x47e;
    long *devset = 0x4c2;

    getbpb = (PFL) *bpbvect; /* save old vectors */
    mediach = (PFW) *mcvect;
    rwabs = (PFL) *rwvect;
    *bpbvect = RDgetbpb; /* install new ones */
    *mcvect = RDmediach;
    *rwvect = RDrwabs;

    /* vectors set-up, include in deviceset : */
    *devset = (*devset) | (0x8L);
}

sleep() /* sleep forever */
{
    int i;
    while (1)
    {
        i = evtnt_timer(30000,0); /* wait 30 Sec. */
        Bconout(2,7); /* BEEP to show I'm alive */
    }
}

main()
{
    appl_init(); /* this is needed even to link - Yack !!!!!!! */

    Rwabs(0,data,720,0,0); /* copy drive A: into RAMdisk data array */

    xbios(38,install); /* INSTALL vectors in SUPV MODE */
    sleep(); /* accessories never end ..... */
}

```

(continued)

St.doc

TEXT

"Programming Tools and the Atari 520ST", Bruce Webster.
February, page 331. Also download ST2.doc.

atari/tech.st #239, from bwebster, 14398 chars, Thu Jan 30 23:46:40 1986

TITLE: *** ALERT! ALERT! ALERT! ***

Checked into my arpa/uucp mail node for the first time in about a month and found the following message. Could make for some interesting complaints about the Feb BEST OF BIX section.

This is a modification to gert's original posting. You should read it carefully if you are planning to upgrade your ST as there are some important modifications. I got this from a posting on net.micro.atari.

>From bammi@cwruecmp.UUCP (Jwahar R. Bammi) Wed Dec 18 13:09:35 1985
>Path: umcp-cs!seismo!harvard!think!mit-eddie!genrad!decvax!cwruecmp!bammi
>Newsgroups: net.micro.atari
>Subject: Re: one meg upgrades - PLEASE READ or fry your ST
>References: <157@imagen.UUCP>

> ~~~~~pop goes the ST~~~~~
>
> i just fried my ST and after talking to atari and my local dealer i
> have tracked down the problem. It is related to the 1 Meg upgrade
> and the new proms, it seems that the early postings of "how to
> upgrade" left out 4 critical resistors (60 ohm 10% tol); 2 must
> be placed on the CAS lines and 2 on the RAS lines, all 4 go on the
> MMU (i know it isnt really an MMU) side. the upgrade will work fine
> w/o the resistors until you put in the new proms, the difference in
> the current drain will cost you all your memory chips and possibly
> the MMU.
> --
>
> god bless Lily St. Cyr
> -Rocky Horror Picture Show
>
> Name: James Turner
> Mail: Imagen Corp. 2650 San Tomas Expressway, P.O. Box 58101
> Santa Clara, CA 95052-9400
> AT&T: (408) 986-9400
> UUCP: ...{decvax,ucbvax}!decwrl!imagen!turner

After reading the above article, I downloaded the revised procedure from Compu Serve. I have had the upgrade for about 3 months, and has worked without a flaw. I would have probably added the proms without the resistors. Thanks to Mr. Turner for the warning.
Here is a copy of the revised procedure:

NOTE: This is an REVISED,TESTED version of the original text downloaded from CompuServe. December 6, 1985

(This was REVISED AND TESTED by an anonymous engineer on Atari's development staff. The addition of the resistors should provide a long life to your machine, but the warning below is STILL IN EFFECT. This is not an official sanction of the modification. USE WITH CARE!!!)

Here's the 1 Meg upgrade directions:

I have brought this over un-editted from the arpanet info-st mailing list. I TAKE NO RESPONSIBILITY FOR ITS CONTENT OR ACCURACY. I HAVE NOT TRIED THIS MODIFICATION ON MY OWN ST AS YET. I AM PASSING THIS ALONG TO THOSE WHO DO WISH TO TRY IT. FOLLOW THE DIRECTIONS AT YOUR OWN RISK.
--Dwight McKay (75776,1521)

From: gert@pescadero

WARNING: This is a hardware modification that will void the warranty of your 520ST. If you do not have the appropriate tools or experience you have a substantial chance of ruining your 520ST. Proceed at your own risk! This modification has been in my 520ST without any problems for 6 days now. However, I have (of course) not checked with knowledgeable sources at Atari to verify if this modification endangers the long term machine reliability and/or software compatibility (I suspect it may endanger their software compatibility if enough of us do it!)

Tools & components needed :

16 256k * 1 RAM chips, 150 ns access time type, e. g. NEC 41256C-15 (available at e. g. Fry's Electronics, Sunnyvale, CA for \$2.77 each)

A good quality, preferably temperature controlled soldering iron, with a miniature tip (tip should be narrow enough to avoid touching 2 I. C. pins at the same time). E. g. Weller type soldering station.

Good quality resin core solder (thin).

Approximately 4 foot of #24 AWG insulated wire and a good stripper for it and 2 feet of #22 AWG solid tinned copper bus wire. You will have to route 3 wires over a sequence if I.C. pins.

Desoldering wick and solder suction tool.

Philips typ e screwdriver (for opening your ST), tweezers, pliers, etc.

A steady hand and self-confidence.

Explanation of the modification :

(Please read the rest of this document before starting. It may save you time and an 520ST)

The current memory inside the 530ST consists of 16 256K*1 RAM chips. Address (A0..A8) lines are common to all those chips. The WriteEnable line is also common to all chips. Data (in and out) lines are of course individual. The RAS (row-address strobe) line is common to all chips. The 8 chips forming the high order byte group have one common CAS line, and the 8 forming the low order byte group have one common CAS line (CAS is used as enable for write operations, such that WriteEnable can be common to both groups). The high order group from MSB to LSB consists of U45, 44, 43, 42, 38, 34, 33, 32. The low order group of U30, 29, 28, 25, 24, 28, 27, 26. Note that all chips are adjacent, though the numbering has gaps. RAS0, CAS0H, and CAS0L are supplied from U1 pin 8, 6 and 7 respectively (The 0 indicates bank 0)

Bank 1 that you are going to build in will be "piggy-backed" on top of the current chips, where all pins of the new chips EXCEPT RAS (pin 4) and CAS (pin 15) are soldered to the old chips equivalent pins. Thus they will end up sharing addresses, data, WriteEnable and power and ground with the existing chips.

All RAS pins of the new chips are wired together and will be supplied with the "RAS1" signal generated on pin 18 of U15 (the memory controller, marked 3H-2119C or so). The CAS pins of the 8 new high order byte chips (on top of U45..U32) are wired together and supplied from the "CAS1H" signal generated on pin 22 of U15. Analogously, the CAS pins of the new U30 to U16 are wired together and supplied with "CAS1L" from pin 21 of U15.

How to go about it:

Step 1: Open up your 520ST, pull off the keyboard connector

(continued)

and remove the main circuit card from its top and bottom shielding. Make sure to remember which screws go where and note the keyboard connector orientation.

Step 2: Desolder all of the capacitors adjacent to the existing RAM chips. (DO NOT SKIP THIS STEP. You'll lose time if you do, and worse, the modification will no be reliable since you can't solder pins obstructed by the capacitors reliably (if at all)). To desolder them, I found it easiest to heat the island on the non component side, and bend the wires straight. After doing that on each capacitor, turn over to the component side and heat the islands while pulling the capacitor out with the tweezers.

Step 3: Open up the holes of all the desoldered capacitors, using a combination of de-soldering wick and suction tool. Do this from the non component side. If certain holes are difficult to open up, you may want to use a wood splinter. (push it through while heating). Be careful to remove all solder debris!! THE REASON for opening the holes NOW is that they will be less accessible once you've done the other steps! Patience is a virtue.

(NOTE: Step 2 & 3 are the only ones that may damage your ST PC board. Be sure not to use excessive force while pulling out the capacitors. If you damage your PC board anyway, cure the problem now and not later).

Step 4: In this step we will piggyback the new RAM's on top of the old ones. Be sure to connect all pins except pin 4 (RAS) and 15 (CAS). The best way to go about this is to do chip by chip. First, bend the pins of the new RAM's such that they are perpendicular to the package (instead of having slightly spread "cowboy legs"). Use pliers to bend pin 4 and 15 such that the legs are 180 degrees from their normal position, so they stick up in the air above the plane of the top surface of the chips. Don't make an absolute sharp 180 degree bend since some manufacturers' pins may snap off. Leave a little curve in the leg, but insure that it is above the plane of the top surface of the chip.

Using #22 AWG to #16 AWG tinned solid copper wire you will form three buses along the top surface of the new d-rams. Cut a #22 AWG solid copper wire the length of the 16 d-rams on the PCB. The RAS bus is formed by soldering all the pin 4's of the new d-rams to the solid copper wire. The bus wire must be seated against the top surface of the new d-rams without a gap. This insures clearance between the top shield and the pins of the d-rams.

After soldering all 16 d-rams to the bus clip off any portion of the pins that extend above the top of the bus wire. Now cut a #22 AWG solid copper wire the length of the 16 d-rams. Place the bus wire along the top surface of the new d-rams in contact with all the pin 15's of the new d-rams. Solder every pin 15 to this bus and as above insure that the wire is seated solidly against the top surface of the new d-rams. Cut off all excess pin length sticking up above the top of the bus wire. Using diagonal cutters remove the section of the bus connecting the new U30 pin 15 to the new U32 pin 15. This divides the bus in half with the new U16, 17, 18, 24, 28, 29 having a common pin 15. The new U32, 33, 34, 38, 42, 43, 44, 45 now have a common pin 15, separated from the other common bus.

(NOTE: until step 6 is finished, do not in any way apply power to your ST. This intermediate state of affairs will damage your memory chips!!)

Step 5: Remount all the desoldered capacitors. Bend the pins like they were before resoldering, such that they will not touch the lower shielding. Solder from the non component side.

Step 6: Orient the 520ST PCB so that you are looking at the solder side of the PCB (non-component side), with the row of

d-rams nearest you. Find the double square pattern of pads at the 68-pin socket of the memory controller, U15 (3H2119). The following is a guide to locating the six memory controller pins necessary to complete the wiring. The socket is numbered counterclockwise, starting with pin 1, the square pad (look closely) in the middle of the bottom outside row. The sequence, moving counterclockwise from pin 1, first on the outside square ONLY: (NOTE: the sequence ")(" means to make a 90-degree turn counterclockwise, i.e. around the corner)

1,3,5,7,9)(10,12,14,16,18,20,22,24,26)(27,29,31,33,35,37,39,41,43)(44,46,48,50,52,54,56,58,60)(61,63,65,67

The sequence, moving counterclockwise along the inside square only, and starting with the left side of the bottom row:

(62,64,66,68,2,4,6,8)(11,13,15,17,19,21,23,25)(28,30,32,34,36,38,40,42)(45,47,49,51,53,55,57,59)

Six 68-ohm 1/4W plus/minus 10% carbon film resistors must be added when adding memory. These series terminating resistors minimize undershoot which may damage BOTH BANKS of d-rams if omitted. Solder a 68-ohm resistor to pin 18 of U15, RAS1. Solder a #24 AWG stranded wire from the remaining end of the 68-ohm resistor to the pin 4 bus (RAS) of all the new d-rams, that is the new U16, 17, 18, 24, 25, 28, 29, 30, 32, 33, 34, 38, 42, 43, 44, and 45.

Solder a 68-ohm resistor to pin 22 of U15, CASH1. Solder a #24 AWG stranded wire from the remaining end of the 68-ohm resistor to pin 15 bus (CAS) of the new U45,44,43,42,38,34,33,32.

Solder a 68-ohm resistor to pin 21 of U15, CASIL. Solder a #24 AWG stranded wire from the remaining end of the 68-ohm resistor to pin 15 bus (CAS) of the new U30, 29,28,25,24,18,17,16.

For best results in all three cases above solder the wires coming from the resistors to the middle of the three bus wires in a "T" fashion rather than at one end of the buses.

Use a continuity tester to find the following three traces -- do not depend on visual inspection. Now install three 68-ohm series terminating resistors in the original 512K bank of ram. Be very careful while soldering to these narrow traces, since excessive heat can easily lift a trace from the board. Use an Exacto knife to gently remove solder mask from traces.

Cut the trace leading from pin 8, RAS0, of U15 near U15. Solder a 68-ohm resistor in series with the trace.

Cut the trace leading from pin 6, CAS0H, of U15 near U15. Solder a 68-ohm resistor in series with the trace.

Cut the trace leading from pin 7, CAS0L, of U15 near U15. Solder a 68-ohm resistor in series with the trace.

Step 7: Sit back. Use Brain. Do you feel confident about the quality of your work? No mistakes? Check evrything once again if you are but a little uncertain. Applying power with errors might make your ST into a decorative, nonfunctional piece of art. OK. Either rebuild your ST into its shielding and cabinet, or put it onto a surface clear of wires and solder remians and connect it to monitor, disk and supply. Boot it.

It it boots, you're probably there. Test if the new memory works by looking at the phystop variable (\$42E) with SID if you have the developer stuff. It should read \$100000 (1M hex). Also note that memcntl (424) now holds 5 instead of 4, and that v_bas_ad (\$44E) now holds \$F80000 (screen bitmap

(continued)

origin). If you don't have the developer stuff, try a single drive copy and check that you get the whole disk in one buffer instead of two.

If the new memory does not seem to exist, use SID to deposit and retrieve words on locations \$80000 and up (1/2 Meg hex). If bit errors occur, the ST bootROM did not detect the extension (it checks all bits of 512 locations by testing a pseudo random sequence, before accepting a memory bank). Try to pin point the faulty chip(s) and remove the error.

If it doesn't boot, you're in trouble. I'm sorry. It is difficult to give hints on what to do here. So many possibilities. Desoldering the new chips probably won't work (if the old ones were functional, the ST would still boot). Check for hidden short:circuit on the RAM pins. May also be that you have a flaky new pin connection.

That's all there is...

--

Jwahar R. Bammi
 Usenet:!decvax!cwruecmp!bammi
 CSnet: bammi@case
 Arpa: bammi%case@csnet-relay
 CompuServe: 71515,155

gimpel.lbr

BINARY

"Processing Strings in Snobol4," James F. Gimpel.
 February, page 175. Download lu.exe to unpack this library.

```

      LOWS = 'abcdefghijklmnopqrstuvwxyz'
      UPS  = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
      INPUT( 'NAMES', 2 )
      PATTERN = TRIM(NAMES)
LOOP1  PATTERN = PATTERN | TRIM(NAMES)          :S(LOOP1)
      PATTERN = PATTERN . NM
LOOP2  LINE = INPUT                             :F(END)
LOOP3  LINE PATTERN = REPLACE(NM, LOWS, UPS )   :S(LOOP3)
      OUTPUT = LINE                           :(LOOP2)
END

```

LISTING2.TXT

```

      DEFINE( 'ROMAN(N)T' )                      :(ROMAN_END)
ROMAN  N RPOS(1) LEN(1) . T                      :F(RETURN)
      '0,1I,2II,3III,4IV,5V,6VI,7VII,8VIII,9IX,'
+      T BREAK(',') . T
      ROMAN = REPLACE( ROMAN(N), 'IVXLCDM', 'XLCDM**') T
+      :S(RETURN)F(FRETURN)
ROMAN_END

```

LISTING3.TXT

```

      DEFINE ( 'SELECT(S)N' )                      :(SELECT_END)
SELECT  S RPOS(1) LEN(1) . N
      N = RANDOM(N)
      S (N-1) ARB . SELECT N                      :(RETURN)
SELECT_END
      DEFS = TABLE()
      DEFS[ 'SENT' ] = '0The <NOUN> <VERB>s the <NOUN>1'
      DEFS[ 'NOUN' ] = '0boy1man2dog3<NOUN> who <VERB>s the <NOUN>4'
      DEFS[ 'VERB' ] = '0bite1walk2pet3lick4smack5'
      STACK = '<SENT>'
      SENTENCE =
L1      STACK POS(0) '<' BREAK('>') . NM '>' =      :F(L2)
      STACK = SELECT( DEFS[NM] ) STACK              :(L1)
L2      STACK BREAK('<') .S =                        :F(L3)

```



```

      SENTENCE = SENTENCE S           :(L1)
L3    SENTENCE = SENTENCE STACK
      .
      .
      .

LISTING4.TXT

      DEFINE( 'PUSH(X)' )
      DEFINE( 'POP()' )
      DEFINE( 'TOP()' )
      DATA( 'LINK(NEXT,VALUE)' )      :(STACK_END)

PUSH  PUSH_POP = LINK( PUSH_POP, X )
      PUSH = .VALUE( PUSH_POP )        :(NRETURN)

POP   IDENT( PUSH_POP )                :S(FRETURN)
      POP = VALUE(PUSH_POP)
      PUSH_POP = NEXT( PUSH_POP )      :(RETURN)

TOP   IDENT( PUSH_POP )                :S(FRETURN)
      TOP = .VALUE( PUSH_POP )         :(NRETURN)

STACK_END

```

LISTING5.TXT

```

      LET = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
      DIGITS = '0123456789'
      IDEN = (ANY(LET) (SPAN(LET DIGITS) | '')) . *PUSH(
+      NULL . *EV()
      INTEGER = SPAN(DIGITS) . *PUSH(
      PRIMARY = IDEN | INTEGER | '(' *E ')'
      FACTOR = PRIMARY | '-' PRIMARY . *NEG()
      TERM = FACTOR
+      ARBNO( '*' FACTOR . *MUL() | '/' FACTOR . *DIV() )
      E = TERM ARBNO( '+' TERM . *ADD() | '-' TERM . *SUB() )

```

dvorak.bas

TEXT

"Keyboard Efficiency," Donald W. Olson and Laurie E. Jasinski.
February, page 241. Also download dvorak.txt.

```

100 TEXT : HOME : VTAB 10
110 PRINT "Q W E R T Y VS. D V O R A K"
120 PRINT
130 PRINT "BY D.W.OLSON AND L.E.JASINSKI"
140 DIM LA$(2),LI$(200),CH$(40),AA(40)
150 DIM FI(122,2),X(122,2),Y(122,2),SK(122,2)
160 DIM CX(8,2),CY(8,2),DS(4,6)
170 DIM FT(8,2),HT(2,2),GT(2)
180 GOSUB 1000:SP = - 16336
200 REM MENU
210 HOME : VTAB 10
220 PRINT "1..START NEW FILE (ZEROS TOTALS)"
230 PRINT : PRINT "2..ADD TO CURRENT FILE"
240 PRINT : PRINT "3..PRINT HARD COPY"
250 PRINT : INPUT A
260 ON A GOTO 300,400,900: GOTO 200
300 REM START NEW FILE
310 NL = 0:NW = 0:AA(0) = 32:CL = 1
320 FOR K = 1 TO 2: FOR F = 1 TO 8
330 FT(F,K) = 0:CX(F,K) = 0:CY(F,K) = 0
340 NEXT : NEXT
350 HOME : GOSUB 500: GOTO 600
400 REM ADD TO CURRENT FILE
410 HOME : VTAB 23: PRINT LI$(NL)
420 PRINT : GOSUB 500: GOTO 600
500 REM PRINT TOTALS

```

```

510 HTAB 1: FOR I = 1 TO 12
515 VTAB I: CALL - 868: NEXT
520 VTAB 1: HTAB 26: PRINT "<ESC: MENU>"
530 PRINT "AFTER ";NW;" WORDS: ": PRINT
540 FOR K = 1 TO 2
550 HT(1,K) = INT (0.5 + FT(1,K) + FT(2,K) + FT(3,K) + FT(4,K))
555 HT(2,K) = INT (0.5 + FT(5,K) + FT(6,K) + FT(7,K) + FT(8,K))
560 GT(K) = HT(1,K) + HT(2,K)
570 INVERSE : PRINT LA$(K);"- TOTAL INCHES: ";GT(K)
580 NORMAL : PRINT "LEFT HAND: ";HT(1,K); SPC( 5 - LEN ( STR$(HT(1,K))));
585 PRINT "      RIGHT HAND: ";HT(2,K)
590 FOR F = 1 TO 8:AX = INT (0.5 + FT(F,K)): PRINT AX; SPC( 5 - LEN ( STR$ (AX)));
595 NEXT F: PRINT : NEXT K: RETURN
600 REM GET NEXT LINE
610 NL = NL + 1: POKE - 16368,0
620 VTAB 23: HTAB 39: PRINT "|";
625 HTAB 1: PRINT CHR$ (7);
630 FOR J = 1 TO 38
635 GET CH$(J):AC = ASC (CH$(J)):AA(J) = AC
640 IF AC > 31 AND AC < 123 THEN 670
645 IF AC = 13 THEN LL = J: GOTO 680
650 IF AC = 8 AND J > 1 THEN PRINT CHR$ (8); CHR$ (32); CHR$ (8);:J = J - 1: GOTO 635
655 IF AC = 8 AND J = 1 THEN GOTO 635
660 IF AC = 27 THEN NL = NL - 1: GOTO 200
665 GOTO 635
670 PRINT CH$(J);: IF J > 31 THEN PRINT CHR$ (7);
675 NEXT J:LL = 39
680 CH$(LL) = " ":AA(LL) = 32
685 LI$(NL) = "": FOR J = 1 TO LL
690 LI$(NL) = LI$(NL) + CH$(J): NEXT
700 REM ANALYZE LINE NUMBER NL
710 HTAB 1: CALL - 868
720 FOR J = 1 TO LL
730 PRINT CH$(J);:AC = AA(J): IF AC < > 32 THEN AX = PEEK (SP) + PEEK (SP)
735 IF AC > 96 THEN CL = 0
740 IF AA(J) = 32 AND AA(J - 1) < > 32 THEN NW = NW + 1
745 IF AA(J) = 32 AND AA(J - 1) = 45 THEN NW = NW - 1
750 FOR K = 1 TO 2:F = FI(AC,K)
755 II = X(AC,K) - CX(F,K) + 2
760 JJ = Y(AC,K) - CY(F,K) + 3
765 CX(F,K) = X(AC,K):CY(F,K) = Y(AC,K)
770 FT(F,K) = FT(F,K) + DS(II,JJ)
775 SF = SK(AC,K): IF CL = 1 AND AC > 64 AND AC < 91 THEN 790
780 IF SF > 0 AND SK(AA(J - 1),K) < > SF THEN FT(SF,K) = FT(SF,K) + 2.25
790 NEXT : NEXT : PRINT : PRINT
795 GOSUB 810: GOSUB 500: GOTO 600
800 REM FINGERS TO HOME ROW
810 FOR K = 1 TO 2: FOR F = 1 TO 8
820 IF CX(F,K) = 0 AND CY(F,K) = 0 THEN GOTO 860
830 II = 2 - CX(F,K):JJ = 3 - CY(F,K)
840 FT(F,K) = FT(F,K) + DS(II,JJ)
850 CX(F,K) = 0:CY(F,K) = 0
860 NEXT : NEXT : RETURN
900 REM PRINT HARD COPY
910 IF NL = 0 THEN 200
920 PRINT : PRINT CHR$ (4);"PR#1"
930 PRINT : PRINT : GOSUB 530: PRINT : PRINT
940 FOR I = 1 TO NL: PRINT LI$(I): NEXT I
950 PRINT : PRINT CHR$ (4);"PR#0"
960 GOTO 200
1000 REM KEYBOARD DATA
1005 LA$(1) = "QWERTY ":LA$(2) = "DVORAK "
1010 READ AC: IF AC = 999 THEN 1200
1015 READ N$: FOR K = 1 TO 2
1020 READ FI(AC,K),X(AC,K),Y(AC,K),SK(AC,K)
1025 NEXT K: GOTO 1010
1032 DATA 32,SPACE,0,0,0,0,0,0,0,0
1033 DATA 33,! ,1,0,2,8,1,0,2,8
1034 DATA 34,QUOTES,8,1,0,1,1,0,1,8
1035 DATA 35,#,3,0,2,8,3,0,2,8
1036 DATA 36,$,4,0,2,8,4,0,2,8
1037 DATA 37,%,4,1,2,8,4,1,2,8
1038 DATA 38,&,5,0,2,1,5,0,2,1
1039 DATA 39,APOSTROPHE,8,1,0,0,1,0,1,0
1040 DATA 40,(,7,0,2,1,7,0,2,1
1041 DATA 41,),8,0,2,1,8,0,2,1

```



```

1042 DATA 42,*,6,0,2,1,6,0,2,1
1043 DATA 43,+,8,2,2,1,8,2,1,1
1044 DATA 44,COMMA,6,0,-1,0,2,0,1,0
1045 DATA 45,-,8,1,2,0,8,1,0,0
1046 DATA 46,.,7,0,-1,0,3,0,1,0
1047 DATA 47,/,8,0,-1,0,8,1,1,0
1048 DATA 48,0,8,0,2,0,8,0,2,0
1049 DATA 49,1,1,0,2,0,1,0,2,0
1050 DATA 50,2,2,0,2,0,2,0,2,0
1051 DATA 51,3,3,0,2,0,3,0,2,0
1052 DATA 52,4,4,0,2,0,4,0,2,0
1053 DATA 53,5,4,1,2,0,4,1,2,0
1054 DATA 54,6,5,-1,2,0,5,-1,2,0
1055 DATA 55,7,5,0,2,0,5,0,2,0
1056 DATA 56,8,6,0,2,0,6,0,2,0
1057 DATA 57,9,7,0,2,0,7,0,2,0
1058 DATA 58, COLON,8,0,0,1,1,0,-1,8
1059 DATA 59, SEMICOLON,8,0,0,0,1,0,-1,0
1060 DATA 60,<,6,0,-1,1,2,0,1,8
1061 DATA 61,=,8,2,2,0,8,2,1,0
1062 DATA 62,>,7,0,-1,1,3,0,1,8
1063 DATA 63,?,8,0,-1,1,8,1,1,1
1064 DATA 64,@,2,0,2,8,2,0,2,8
1065 DATA 65,A,1,0,0,8,1,0,0,8
1066 DATA 66,B,4,1,-1,8,5,-1,-1,1
1067 DATA 67,C,3,0,-1,8,6,0,1,1
1068 DATA 68,D,3,0,0,8,5,-1,0,1
1069 DATA 69,E,3,0,1,8,3,0,0,8
1070 DATA 70,F,4,0,0,8,5,-1,1,1
1071 DATA 71,G,4,1,0,8,5,0,1,1
1072 DATA 72,H,5,-1,0,1,5,0,0,1
1073 DATA 73,I,6,0,1,1,4,1,0,8
1074 DATA 74,J,5,0,0,1,3,0,-1,8
1075 DATA 75,K,6,0,0,1,4,0,-1,8
1076 DATA 76,L,7,0,0,1,8,0,1,1
1077 DATA 77,M,5,0,-1,1,5,0,-1,1
1078 DATA 78,N,5,-1,-1,1,7,0,0,1
1079 DATA 79,O,7,0,1,1,2,0,0,8
1080 DATA 80,P,8,0,1,1,4,0,1,8
1081 DATA 81,Q,1,0,1,8,2,0,-1,8
1082 DATA 82,R,4,0,1,8,7,0,1,1
1083 DATA 83,S,2,0,0,8,8,0,0,1
1084 DATA 84,T,4,1,1,8,6,0,0,1
1085 DATA 85,U,5,0,1,1,4,0,0,8
1086 DATA 86,V,4,0,-1,8,7,0,-1,1
1087 DATA 87,W,2,0,1,8,6,0,-1,1
1088 DATA 88,X,2,0,-1,8,4,1,-1,8
1089 DATA 89,Y,5,-1,1,1,4,1,1,8
1090 DATA 90,Z,1,0,-1,8,8,0,-1,1
1091 DATA 91,[,8,1,1,0,8,1,2,0
1092 DATA 92,\,8,3,1,0,8,3,1,0
1093 DATA 93,],8,2,1,0,8,2,2,0
1094 DATA 94,^,5,-1,2,1,5,-1,2,1
1095 DATA 95,_,8,1,2,1,8,1,0,1
1096 DATA 96,',1,-2,-2,0,1,-2,-2,0
1123 DATA 999
1200 FOR I = 65 TO 90:AC = I + 32
1210 FOR K = 1 TO 2
1220 FI(AC,K) = FI(I,K):SK(AC,K) = 0
1230 X(AC,K) = X(I,K):Y(AC,K) = Y(I,K)
1240 NEXT K: NEXT I
1300 REM KEYBOARD SPACINGS
1310 U = .75:V = .81:AN = 90 + 23
1320 DP = U * V * COS (AN * 3.1416 / 180)
1330 FOR I = 0 TO 4: FOR J = 0 TO 6
1340 DX = I - 2:DY = J - 3
1350 DS(I,J) = SQR (DX * DX * U * U + DY * DY * V * V + 2 * DX * DY * DP)
1360 NEXT J: NEXT I: RETURN

```

(continued)

```

program puzzle;
(* A compute-bound program from Forest Baskett *)

const
  size      = 511;
  classMax  = 3;
  typeMax   = 12;
  d         = 8;
type
  pieceClass = 0..classMax;
  pieceType  = 0..typeMax;
  position   = 0..size;
var
  pieceCount : array [pieceClass] of 0..13;
  class       : array [pieceType] of pieceClass;
  pieceMax    : array [pieceType] of position;
  puzzle      : array [position] of boolean;
  p           : array [pieceType, position] of boolean;
  m,n         : position;
  i,j,k       : 0..13;
  kount       : integer;
function fit (i : pieceType; j : position) : boolean;
label 1;
var
  k : position;
begin fit := false; for k := 0 to pieceMax[i] do
  if p[i,k] then if puzzle[j+k] then goto 1;
fit := true;
1:
end;

function place (i : pieceType; j : position) : position;
label 1;
var
  k : position;
begin
  for k := 0 to pieceMax[i] do
    if p[i,k] then puzzle[j+k] := true;
    pieceCount[class[i]] := pieceCount[class[i]] - 1;
    for k := j to size do
      if not puzzle[k] then begin
        place := k;
        goto 1;
      end;
  writeln('Puzzle filled.');
```

```

  place := 0;
1:
end;
procedure remove (i : pieceType; j : position);
var k : position;
begin
  for k := 0 to pieceMax[i] do
    if p[i,k] then puzzle[j+k] := false;
    pieceCount[class[i]] := pieceCount[class[i]] + 1;
end;

function trial (j : position) : boolean;
label 1;
var
  i : pieceType;
  k : position;
begin
  for i := 0 to typeMax do
    if pieceCount[class[i]] <> 0 then
      if fit (i,j) then begin
        k := place (i,j);
        if trial(k) or (k = 0) then begin
          writeln('Piece ',i+1,' at ',k+1);
          trial := true;

```



```

        goto 1;
    end else remove (i,j);
end;
trial := false;
1:kount := kount + 1;
end;
begin
    writeln('Beginning PUZZLE');
    for m := 0 to size do puzzle[m] := true;
    for i := 1 to 5 do for j := 1 to 5 do for k := 1 to 5 do
        puzzle[i+d*(j+d*k)] := false;
    for i := 0 to typeMax do for m := 0 to size do p[i,m] := false;
    for i := 0 to 3 do for j := 0 to 1 do for k := 0 to 0 do
        p[0,i+d*(j+d*k)] := true;

class[0] := 0;
pieceMax[0] := 3+d*1+d*d*0;
for i := 0 to 1 do for j := 0 to 0 do for k := 0 to 3 do
    p[1,i+d*(j+d*k)] := true;

class[1] := 0;
pieceMax[1] := 1+d*0+d*d*3;
for i := 0 to 0 do for j := 0 to 3 do for k := 0 to 1 do
    p[2,i+d*(j+d*k)] := true;

class[2] := 0;
pieceMax[2] := 0+d*3+d*d*1;
for i := 0 to 1 do for j := 0 to 3 do for k := 0 to 0 do
    p[3,i+d*(j+d*k)] := true;

class[3] := 0;
pieceMax[3] := 1+d*3+d*d*0;
for i := 0 to 3 do for j := 0 to 0 do for k := 0 to 1 do
    p[4,i+d*(j+d*k)] := true;

class[4] := 0;
pieceMax[4] := 3+d*0+d*d*1;
for i := 0 to 0 do for j := 0 to 1 do for k := 0 to 3 do
    p[5,i+d*(j+d*k)] := true;

class[5] := 0;
pieceMax[5] := 0+d*1+d*d*3;
for i := 0 to 2 do for j := 0 to 0 do for k := 0 to 0 do
    p[6,i+d*(j+d*k)] := true;

class[6] := 1;
pieceMax[6] := 2+d*0+d*d*0;
for i := 0 to 0 do for j := 0 to 2 do for k := 0 to 0 do
    p[7,i+d*(j+d*k)] := true;

class[7] := 1;
pieceMax[7] := 0+d*2+d*d*0;
for i := 0 to 0 do for j := 0 to 0 do for k := 0 to 2 do
    p[8,i+d*(j+d*k)] := true;

class[8] := 1;
pieceMax[8] := 0+d*0+d*d*2;
for i := 0 to 1 do for j := 0 to 1 do for k := 0 to 0 do
    p[9,i+d*(j+d*k)] := true;

class[9] := 2;
pieceMax[9] := 1+d*1+d*d*0;
for i := 0 to 1 do for j := 0 to 0 do for k := 0 to 1 do
    p[10,i+d*(j+d*k)] := true;

class[10] := 2;
pieceMax[10] := 1+d*0+d*d*1;
for i := 0 to 0 do for j := 0 to 1 do for k := 0 to 1 do
    p[11,i+d*(j+d*k)] := true;

class[11] := 2;
pieceMax[11] := 0+d*1+d*d*1;
for i := 0 to 1 do for j := 0 to 1 do for k := 0 to 1 do
    p[12,i+d*(j+d*k)] := true;

```

(continued)

```

class[12] := 3;
pieceMax[12] := 1+d*1+d*d*1;
pieceCount[0] := 13;
pieceCount[1] := 3;
pieceCount[2] := 1;
pieceCount[3] := 1;
m := 1+d*(1+d*1);
kount := 0;
if fit(0,m) then n := place(0,m) else writeln('Error.');
```

```

if trial(n) then writeln('Success in ',kount,' trials.')
else writeln('Failure.');
```

```

end.
```

badfile.c

TEXT

Programming Insight: "Badfile: CP/M System Programming in C," Louis Baker.
February, page 157.

```

/* program to identify which file reside on "bad" track&sector */
/* Copyright 1985 Louis Baker all rights reserved */

#include "libc.h"

#define ESC 27
#define CR 13
#define LF 10
#define FF 255 /* code returned by find bdos call if no file */
#define DFCB 92 /* 92 = 5CH address of default file control blk */
#define DMA 128 /* address of DMA */

struct dpb {
    char spt[2]; /* low order byte first */
    char bsh;
    int blmexm,dsm,drm,al,cks; /*not used */
    char off[2];
} /* disk parameter block structure */ ;

struct fcb{
    char drive;
    char fname[8];
    char type[3];
    char fex;
    char sys[2];
    char frec;
    char falg[16];
    char cr;
    char r0,r1,r2;
} /* file control block */;

main (argc,argv) /* IDENTIFY FILE CORRESP. TO BAD SECTOR */
int argc;
{
    register int i;
    static int mode,alg,track,sector,secpt,offset,bls,length,j;
    static int bad,blksf,driven,bc,de;
    int *hl;
    struct fcb *fcbp,*fcb2;
    struct dpb *dpbp;
    char name[13],byte;

    /* CPM version number */
    bc=12; de=0 /* unused */; j = bdos(bc,de) /* this works */
    printf(" CP/M version number %x\n",j);
    /* desired drive? */
    printf("enter drive (default=0,A=1,B=2,etc) ");
    scanf(" %d",&driven) /* scanf need pointers */;
    /*input desired mode of search */
    printf("enter 0 if track/sector given, 1 if group");
    scanf(" %d",&mode);
    /* BIOS CALL to select disk if not default */
    bc=driven-1; /* bc registers for disk selection */
    /* SELDSK 9th bios entry hl points to disk parameter
```



```

        header */
        if(bc!=-1) hl = bioshl(9,bc,de);
        printf(" alloc. group of disk parameter header %x\n",hl);
        if(mode==1) { /* read in allocation group */
            printf(" enter hex alloc. gp.");
            scanf("%x",&alg);
            /* use hl= adr of disk parameter header to get dp block */
            hl = hl +5; /* 5 words = 10 bytes */
            /* hl now points to dpb address */
            printf(" address containing dpb address %x\n",hl);
            dpbp= *hl; /* dpbp= contents of what hl points to */
            printf(" loc of dpb %x\n",dpbp);
            /*dpbp points to address in dpb field of dph */
        }
        else {
            printf(" enter track(decimal)");
            scanf("%d",&track);
            printf(" enter sector (decimal)");
            scanf("%d",&sector);
            /* determine allocation group */
            /* another way to locate dp block-BDOS CALL */
            bc = 31;
            dpbp = bdoshl(bc,de) ;/* get dpb address. de unused */
            /* now find allocation group */
            secpt = (dpbp->spt[0]) +256 * (dpbp->spt[1]) ;
            printf(" sectors per track %d\n", secpt);
            offset= (dpbp->off[0])+256*(dpbp->off[1]);
            printf(" offset %d\n",offset);
            blksf = dpbp->bsh;
            /* printf(" loc offset %x\n",&(dpbp->off)); */
            printf(" block shift factor %d\n",blksf);
            alg =
                ( (track-offset)*secpt +sector-1 ) >>(blksf);
        } /* END of else clause */
    /* echo check */
    printf(" alloc.gp.= %x\n", alg); /* code working up to here */
    /* now search for that alloc. gp. */ ;
    fcbp = DFCB /* specify file control block */;
    fcbp->drive= driven /* drive name */;
    /* set file name,type,extent to wild card= ? */
    for (i=0;i<8;i++)
        fcbp->fname[i]= '?';
    fcbp->type[0]= '?';fcbp->type[1]='?';fcbp->type[2]='?';
    fcbp->fex= '?' /* we don't use strings, which require \0
        term. */ ;
    /* loop over files max 64 dir. entries in CP/M*/
    length = dpbp->drm;
    printf(" directory length %d entries\n",length);
    for (bc=17,j=0; j<length;j++,bc=18) {
        mode = bdos(bc,fcbp);
        /* DE=fcbp points to fcb. A=directory code
            in variable mode =FF if done else 0 to 3 */
        if (mode==FF)
            goto fini;
        fcb2 = mode*32 + DMA ;/* point to found fcb */
        /* loop over groups in this extent */
        for(i=0;i<16;i++){
            if(fcb2->falg[i]==alg)
                goto found;
            /* we could put here go to next file if falg=0 */
            if (fcb2->falg[i]=='\0') break;
        } /* end of the for loop over extent*/
    } /* end of for loop over directory entries */
fini:  printf(" no user file at that group\n");
        goto term;
found:/* print file name. get size and approx. position */
        j=fcb2->fex;
        printf(" bad record %d of extent %d\n",i+1,j);
        /* BDOS call for record count */
        bc=35;
        fcb2->drive = fcbp->drive;/* move drive i.d. to
            make fcb out of file info in DMA area */
        hl = bdoshl(bc,fcb2); /* CP/M to get record count*/
        /* call to bdos or CPM equivalent, as answer in fcb */
        if ( (fcb2->r2) == 1 ) length = 65536;

```

(continued)

```

        else length= ((int)(fcb2->r0)) + 256*((int)(fcb2->r1));
        printf(" bad file: %d records\n",length);
/* position of bad sector NB- 1 record can be >1 sector in file */
        length=(100*(16*j+i))/length;
        printf(" bad record approx %d percent into file:\n",length);
        pfilen(fcb2);
term:    exit(0);/*return to system, job done */
}

pfilen(fcbp) struct fcb *fcbp;
{
struct fcb *fcb2;
static char pname[9],ptype[4];
register int i;
        fcb2=fcbp;
        pname[8]='\0';ptype[3]='\0';
        /* move i no longer needed for position of bad gp. */
        for (i=0;i<8;i++) pname[i]= fcb2->fname[i];
        for (i=0;i<3;i++) ptype[i]= fcb2->type[i];
        /* terminate string name-eliminate trailing blanks in name */
        for (i=7;i>-1;i--)
                if(pname[i]==' ')pname[i]='\0';
                else break; /* do NOT eliminate embedded blanks */
}
/* output file ID */
        printf ("%s.%s\n",pname,ptype);
}

```

levien.lbr

BINARY

"Visual Programming," Ralph Levien,
February, page 135. Download lu.exe to unpack this library.

The following files accompany the article, "Visual Programming", by Raph Levien, which appeared in the Feb. 1986 issue of BYTE, page 135:

SMALLVSD - This version of the visual syntax editor requires an IBM PC and BYSO LISP, an implementation of LISP available from Levien Instrument Company, POBox 31, McDowell, VA 24459.

XLISPVSD - This version of the visual syntax editor requires an IBM PC and version 1.5d (or later) of XLISP, a public domain implementation of LISP that is available free of charge from BYTENet Listings and the BYTE Information Exchange (BIX).

XLISP15D.EXE - This is an executable version of XLISP, with special modifications that allow it to run XLISPVSD, a version of the visual syntax editor. XLISP and the XLISP version of the visual syntax editor were written by David Betz. (Note that the article incorrectly identifies this version of XLISP as 1.5c.)

FIB - This is the source code for the Fibonacci function described in the article "Visual Programming", and may be used with either the BYSO LISP or XLISP versions of the visual syntax editor.

GRINDEF - This is the source code for the grindex function described in the article "Visual Programming." XLISP does not include this function, so you must load it (as described below) before you can use it in XLISP.

XLISP15.EXE

Using the Visual Syntax Editor with XLISP 1.5d

To use the visual syntax editor with XLISP 1.5d, enter XLISP by typing

XLISP15D

at the MS-DOS prompt. The program will respond

XLISP version 1.5d, Copyright (c) 1985, by David Betz
>

To load the visual syntax editor, type

```
(load "XLISPVSD")
```

and the program will respond

```
;loading "XLISPVSD"  
T
```

From there, you can follow the examples as given in the article, "Visual Programming," except that to load the Fibonacci function, you type

```
(load "fib")
```

and to load the grindex function (described on page 138) you type

```
(load "grindex")
```

Note: there is a typographical error in the article, and the syntax for using grindex to view the definition of a function in the visual editor is

```
(edv (grindex 'function-name))
```

Further information on XLISP is available in the March 1984 BYTE article, "An XLISP Tutorial," by David Betz, page 221.

GRINDEX

```
; (grindex sym) return the function definition associated with a symbol  
(defun grindex (sym)  
  (if (and (symbolp sym)  
           (boundp sym)  
           (consp (symbol-value sym))  
           (consp (car (symbol-value sym)))  
           (consp (cadr (symbol-value sym)))))  
      '(defun ,sym ,(cadr (symbol-value sym)) ,@(caddr (symbol-value sym))))  
  (defun cadr (x)  
    (cadr (car x)))  
  (defun caddr (x)  
    (caddr (car x)))
```

FIB

```
(defun fib (x) (if (< x 2) 1 (+ (fib (- x 1))  
  (fib (- x 2)))))
```

SMALLVSD

```
"BYSO Visual Syntax Editor-Limited Version"  
"Copyright (C) 1985 Raphael L. Levien"  
"For private, non-commercial use only."  
(putprop 'defun 'vsn 'defund)  
(putprop 'quote 'vsn 'quoted)  
(defsetf in ins)  
(defun vsn (l)  
  (let ((he (gensym)))  
    (cls)  
    (vsn1 (grindex l)  
      160)  
    (setc 3360)))  
(defun vsn1 (l p)  
  (if (eq he l)  
      (highlt l p)  
      (if (consp l)  
          (if (get (car l)  
                  'vsn)  
              (funcall (get (car l)  
                          'vsn)  
                l  
                p)  
              (adj (car l)  
                p)
```

```

(vsd4 (cdr l)
(vsd3 (car l)
p))))
(vsd2 l p)))
(defun vsd2 (l p)
(let* ((s (expand l))
(sl (length s)))
(prog1 (setc (- p (+ sl sl)))
(pstring s))))
(defun vsd3 (a p)
(let* ((s (expand a))
(sl (length s))
(b (- p (+ sl sl 4))))
(setc b)
(try 218)
(for i 1 sl 1 (try 196))
(try 191)
(setc (plus 160 b))
(try 179)
(pstring s)
(try 195)
(setc (+ b 320))
(try 192)
(for i 1 sl 1 (try 196))
(try 217)
b))
(defun adj (a p h)
(let* ((s (expand a))
(sl (length s))
(b (- p (+ sl sl -316)))
(ds (array 'char (+ sl 2))))
(aset 179 ds 0)
(aset 179 ds (+ 1 sl))
(for i 1 sl 1 (aset 32 ds i))
(setc (for i b (- (max b (car h))
160)
160
(setc i)
(pstring ds)))
(try 192)
(for i 1 sl 1 (try 196))
(try 217)
(max b (- (cdr h)
160))))
(defun vsd4 (a p)
(do ((l a (cdr l))
(c p))
((null l)
(cons (+ c 160)
p))
(setc (setq c (+ p (if (consp (car l))
156
-4)))))
(try 196)
(try 26)
(setq p (+ (* 160 (/ (vsd1 (car l)
(- p 4))
160))
(remainder p 160)
160))))
(defun defund (l p)
(setc 0)
(msg "Function: " (cadr l)
t
"Variables:")
(if (and (caddr l)
(atom (caddr l)))
(setq l (cdr l)))
(do ((tl (caddr l)
(cdr tl)))
((null tl)
(msg " " (car tl)))
(vsd1 (if (cddddr l)
(cons 'progn (cddddr l))
(cddddr l)))

```



```

p))
(defun quoted (l p)
  (vsd2 (cadr l)
    (+ 160 p)))
(defun highlt (l p)
  (let ((ll (+ (getc 1)
    0))
    (he (gensym)))
    (prog2 (setc (getc 2)
      1)
      (vsd1 l p)
      (setc ll 1))))
(defun in (x y)
  (if (null y)
    x
    (nth (car y)
      (in x (cdr y)))))
(defun ins (z y v)
  (if (null y)
    (setq x v)
    (setf (nth (car y)
      (in z (cdr y)))
      v)))
(defun edv (x)
  (prog (curs com)
    (setq x (subst nil nil x)
      curs
      (if (and (consp x)
        (eq (car x)
          'defun))
        (list (if (and (caddr x)
          (atom (caddr x)))
            4
            3))
        nil))
    (cls)
    a
    (setq cont x)
    (dhlt x)
    (setq com (tyk))
    (if (= (low com)
      27)
      (stoped))
    (if (= (high com)
      72)
      (setf (car curs)
        (- (car curs)
          1)))
    (if (= (high com)
      75)
      (setq curs (cons 1 curs)))
    (if (= (high com)
      77)
      (setq curs (cdr curs)))
    (if (= (high com)
      80)
      (setf (car curs)
        (+ (car curs)
          1)))
    (if (= (low com)
      99)
      (chel))
    (if (= (low com)
      97)
      (addarg))
    (if (= (low com)
      105)
      (inel))
    (if (= (low com)
      100)
      (delel))
    (if (= (low com)
      116)

```

(continued)

```

(testel))
(go a)))
(defun tyk expr nil (setc (getc))
  (car ((bios 22)
        0)))
(defun dhlt (l)
  (let ((he (in l curs)))
    (vsdi l 160)
    (setc 3360)))
(defun chel expr nil (msg "Change to")
  (setf (in x curs)
    (readel "Change to: "))
  (cls))
(defun readel (m)
  (msg " a)tom or f)unction? ")
  (if (= (low (tyk))
        102)
    (progn (msg "function" t m)
      (list (read)))
    (progn (msg "atom" t m)
      (read))))
(defun addarg expr nil (msg "Add argument")
  (setf (cdr (lastcdr (in x curs)))
    (list (readel "Argument: ")))
  (cls))
(defun lastcdr (x)
  (cond ((null x)
        nil)
        ((null (cdr x))
        x)
        (t (lastcdr (cdr x)))))
(defun inel expr nil (when curs (msg "Insert")
  (setf (nthcdr (car curs)
    (in x (cdr curs)))
    (cons (readel "Insert: ")
      (nthcdr (car curs)
        (in x (cdr curs)))))
  (cls)))
(defun delel expr nil (when curs (setf (nthcdr (car curs)
  (in x (cdr curs)))
    (nthcdr (+ (car curs)
      1)
    (in x (cdr curs)))))
  (if (not (nthcdr (car curs)
    (in x (cdr curs))))
    (if (= (car curs)
      1)
      (setq curs (cdr curs))
      (setf (car curs)
        (- (car curs)
          1))))
  (cls)))
(defun testexp (exp)
  (prog (val)
    (setq val (eval exp))
    (msg "Value: ")
    (print val)
    (msg "Press any key to return to editor: ")
    (tyk)
    (cls)))
(defun testel expr nil (if curs (progn (msg "w)hole display or h)ighlighted area? ")
  (if (= (low (tyk))
        104)
    (progn (msg "highlighted area" t)
      (testexp (in x curs)))
    (progn (msg "whole display" t)
      (testexp x))))
  (testexp x)))
(defun stoped expr nil (msg "Are you sure you want to exit the editor? ")
  (if (= (low (tyk))
        121)
    (progn (terpri)
      (return x))
    (cls)))
(defun ask (m)

```



```
(msg m)
(read))
```

XLISPVSD

```
; BYSO Visual Syntax Editor-Limited Version
; Copyright (C) 1985 Raphael L. Levien
; ALL RIGHTS RESERVED
```

```
; Converted for XLISP version 1.5c on the
IBM-PC by David Betz
```

```
(putprop 'defun 'defund 'vsd)
(putprop 'quote 'quoted 'vsd)
```

```
(setq *he* nil)
```

```
(defun vsd (l)
  (let ((old-he *he*))
    (clear)
    (setq *he* (gensym))
    (vsd1 l 160)
    (setq *he* old-he)
    (setc 3360)))
```

```
(defun vsd1 (l p)
  (if (eq *he* l)
    (highlight l p)
    (if (consp l)
      (if (and (symbolp (car l)) (get (car l) 'vsd))
        (funcall (get (car l) 'vsd) l p)
        (adj
         (car l)
         p
         (vsd4 (cdr l) (vsd3 (car l) p))))
      (vsd2 l p))))
```

```
(defun vsd2 (l p)
  (prog1
    (setc (- p (* (flatc l) 2)))
    (princ l)))
```

```
(defun vsd3 (a p)
  (let* ((sl (flatc a))
        (b (- p (+ sl sl 4))))
    (setc b)
    (write-char 218)
    (dotimes (i sl) (write-char 196))
    (write-char 191)
    (setc (+ 160 b))
    (write-char 179)
    (princ a)
    (write-char 195)
    (setc (+ b 320))
    (write-char 192)
    (dotimes (i sl) (write-char 196))
    (write-char 217)
    b))
```

```
(defun adj (a p h)
  (let* ((sl (flatc a))
        (b (- p (+ sl sl -316)))
        (top (- (max b (car h)) 160)))
    (setc (do ((i b (+ i 160)))
              ((> i top) i)
              (setc i)
              (write-char 179)
              (dotimes (i sl) (write-char 32))
              (write-char 179))))
    (write-char 192)
    (dotimes (i sl) (write-char 196))
    (write-char 217)
    (max b (- (cdr h) 160))))
```

```
(defun vsd4 (a p)
  (do ((l a (cdr l))
      (c p))
```

(continued)

```

(null l)
(cons (+ c 160)
p))
(setc (setq c (+ p (if (consp (car l))
                        156
                        -4))))

(write-char 196)
(write-char 26)
(setq p (+ (* 160 (/ (vsd1 (car l)
                        (- p 4))
                    160))
          (rem p 160)
          160))))

(defun defund (l p)
  (setc 0)
  (msg "Function: " (cadr l)
    t
    "Variables:")
  (if (and (nth 2 l)
          (atom (nth 2 l)))
      (setq l (cdr l)))
  (do ((tl (nth 2 l)
          (cdr tl)))
      ((null tl))
    (msg " " (car tl)))
  (vsd1 (if (nthcdr 4 l)
            (cons 'progn (nthcdr 3 l))
            (nth 3 l))
    p))

(defun quoted (l p)
  (vsd2 (cadr l)
    (+ 160 p)))

(defun hight (l p)
  (let ((old-he *he*))
    (let (r)

```

```

(set-inverse t)
(setq *he* (gensym))
(setq r (vsd1 l p))
(setq *he* old-he)
(set-inverse nil)
r)))

(defun in (x y)
  (if (null y)
      x
      (nth (car y)
        (in x (cdr y)))))

(defun ins (z y v)
  (if (null y)
      (setq **x* v)
      (setf (nth (car y)
        (in z (cdr y)))
        v)))

(defun edv (x)
  (prog (com)
    (setq **x* (subst nil nil x)
      *curs*
      (if (and (consp **x*)
              (eq (car **x*)
                'defun))
          (list (if (and (nth 2 **x*)
                        (atom (nth 2 **x*)))
                    4
                    3))
          nil))
    (clear)
    a
    (dhlt **x*)
    (setq com (get-key))
    (if (= com 27) ; escape
        (stoped))
    (if (= com 200) ; up

```



```

(if *curs* (setf (car *curs*)
  (1- (car *curs*))))))
(if (= com 203)      ; left
  (setq *curs* (cons 1 *curs*)))
(if (= com 205)      ; right
  (setq *curs* (cdr *curs*)))
(if (= com 208)      ; down
  (if *curs* (setf (car *curs*)
    (1+ (car *curs*))))))
(if (= com 99)       ; (c)hange
  (chel))
(if (= com 97)       ; (a)dd
  (addarg))
(if (= com 105)      ; (i)nsert
  (inel))
(if (= com 100)      ; (d)elele
  (delel))
(if (= com 116)      ; (t)est
  (testel))
(go a)))

(defun dhlt (l)
  (let ((old-he *he*))
    (setq *he* (in l *curs*))
    (vsl 1 160)
    (setq *he* old-he)
    (setc 3360)))

(defun chel ()
  (msg "Change to")
  (ins ** *curs* (readl "Change to: "))
  (clear))

(defun readel (m)
  (msg " a)tom or f)unction? ")
  (if (= (get-key) 102)
    (progn
      (msg "function" t m)
      (list (read)))

```

```

    (progn
      (msg "atom" t m)
      (read))))

(defun addarg ()
  (msg "Add argument")
  (setf (cdr (last (in ** *curs*)))
    (list (readl "Argument: ")))
  (clear))

(defun inel ()
  (when *curs*
    (msg "Insert")
    (setf (cdr (nthcdr (1- (car *curs*))
      (in ** (cdr *curs*)))))
      (cons (readl "Insert: ")
        (nthcdr (car *curs*)
          (in ** (cdr *curs*)))))
    (clear)))

(defun delel ()
  (when *curs*
    (setf (cdr (nthcdr (1- (car *curs*))
      (in ** (cdr *curs*)))))
      (nthcdr (1+ (car *curs*))
        (in ** (cdr *curs*)))))
    (if (not (nthcdr (car *curs*)
      (in ** (cdr *curs*)))))
      (if (= (car *curs*) 1)
        (setq *curs* (cdr *curs*))
        (setf (car *curs*) (1- (car *curs*))))
      (clear)))

(defun testexp (exp)
  (prog (val)
    (setf val (eval exp))
    (msg "Value: ")

```

(continued)

```

(print val)
(msg "Press any key to return to editor: ")
(get-key)
(clear)))

(defun testel ()
  (if *curs*
    (progn
      (msg "whole display or h)highlighted area? ")
      (if (= (get-key) 104)
        (progn
          (msg "highlighted area" t)
          (testexp (in ** *curs*)))
        (progn
          (msg "whole display" t)
          (testexp **))))
      (testexp **)))
  (defun stoped ()
    (msg "Are you sure you want to exit
the editor? ")
    (if (= (get-key) 121)

(progn
  (terpri)
  (return **))
(clear)))

(defun ask (m)
  (msg m)
  (read))

; functions required for XLISP

(defun setc (p)
  (set-cursor (/ p 160) (rem (/ p 2) 80))
  p)

(defun msg (&rest args)
  (mapcar #'(lambda (x) (if (eq x t) (terpri)
    (princ x))) args))

(expand 1)

```

farrell.lib

BINARY

Programming Insight: "Molecules in Color," John J. Farrell.
February, page 149. Download lu.exe to unpack. For a TEXT file version,
download farrell.lib.

The following programs accompany the article "Molecules in Color", which describes COLOR3D.BAS, a color IBM PC version of MODEL3D.BAS, a Macintosh program that appeared in the February 1985 BYTE.

COLOR3D.BAS - the program, which requires an IBM PC or compatible, and BASICA

The following are data files to be used with COLOR3D.BAS:

B5H9.DAT
B10H14.DAT
C7H7O2N.DAT
NACL.DAT
CR(C6H6).DAT
CRDIBENZ.DAT
BENZENE.DAT

PATTERNS.DAT - This data file generates the patterns that you may use with COLOR3D.BAS

farrell.lib

TEXT

Programming Insight: "Molecules in Color," John J. Farrell.
February, page 149. All the programs in one file.

```

1000 ' ***** COLOR3D.BAS *****
1010 ' Draws a 3D, perspective image of a molecule on IBM PCs with BASICA.
1020 ' For private, noncommercial use only.
1030 ' John J. Farrell *** April 1, 1985
1040 ' Inspired by Earl Kirkland's MODEL3D.BAS for the Mac, BYTE, Feb. 1985.
1050 SCREEN 1 'medium resolution; color
1060 COLOR 0,1 'background = black(0); cyan(1); magenta(2); white(3)
1070 KEY OFF
1080 DEFINT I-N: DEFSNG O-Z: DEFSNG A-G
1090 DIM X(200), Y(200), Z(200), S(200), COL(200),COLPAT(200),TIL$(200)
1100 '
1110 ' Ask for input parameters.
1120 CLS: INPUT "Data file name:", FILE$
1130 INPUT "Azim., polar angles (phi, theta):", PHI, THETA
1140 INPUT "Viewing distance:",VIEWD
1150 INPUT "Size magnitude:",SMAG
1160 SMAG = 1.15*SMAG
1170 ' DISTORT is used later to account for fact that one unit of x
1180 ' on screen (horizontal) is not equal to one unit of z (vertical).
1190 DISTORT = 1.2
1200 ' Convert degrees to radians.
1210 PHI = PHI*3.14159/180!: THETA = THETA*3.14159/180!
1220 CP = COS(PHI): SP = SIN(PHI): CT = COS(THETA): ST = SIN(THETA)
1230 '
1240 OPEN FILE$ FOR INPUT AS #1
1250 ' Set xmin very large and xmax very small.
1260 XMIN = 1000000!: XMAX = -XMIN: YMIN = XMIN: YMAX = XMAX
1270 ZMIN = XMIN: ZMAX = XMAX: N = 0
1280 ' Read data file: color, x,y,z (atomic coords),r (Angstroms).
1290 WHILE NOT EOF(1)
1300 N = N + 1
1310 INPUT #1,COLPAT(N), X(N),Y(N), Z(N), S(N)
1320 IF COLPAT(N)<= 3 THEN COL(N) = COLPAT(N): TIL$(N) = CHR$(&HAA)
1330 IF COLPAT(N) = 4 THEN COL(N) = 1: TIL$(N) =CHR$(&H66) + CHR$(&H99)
1340 IF COLPAT(N) = 5 THEN COL(N) = 3: TIL$(N) = CHR$(&HAF) +CHR$(&HAF) +
CHR$(&HFA) + CHR$(&HFA)
1350 IF COLPAT(N) = 6 THEN COL(N) = 2: TIL$(N) =CHR$(&H55) + CHR$(&HFF)
1360 IF COLPAT(N) = 7 THEN COL(N) = 3: TIL$(N) = CHR$(&HAA) + CHR$(&H69) +
CHR$(&HFF) + CHR$(&H5A) + CHR$(&HA5) + CHR$(&HFF) + CHR$(&H96) + CHR$(&HAA)
1370 IF COLPAT(N) = 8 THEN COL(N) = 3: TIL$(N) = CHR$(&H5A) + CHR$(&H5A) +
CHR$(&HA5) + CHR$(&HA5)
1380 IF COLPAT(N) = 9 THEN COL(N) = 3: TIL$(N) = CHR$(&HAA) + CHR$(&HAA) +
CHR$(&H55) + CHR$(&H55)
1390 IF COLPAT(N) = 10 THEN COL(N) = 3: TIL$(N) = CHR$(&HAA) + CHR$(&HFF)
1400 IF COLPAT(N) = 11 THEN COL(N) = 3: TIL$(N) = CHR$(&H5F) + CHR$(&H5F) +
CHR$(&HF5) + CHR$(&HF5)
1410 IF COLPAT(N) = 12 THEN COL(N) = 3: TIL$(N) = CHR$(&H69) + CHR$(&HAA) +
CHR$(&HAA) + CHR$(&H96)
1420 IF COLPAT(N) = 13 THEN COL(N) = 3: TIL$(N) = CHR$(&HBB)
1430 IF COLPAT(N) = 14 THEN COL(N) = 3: TIL$(N) = CHR$(&HAB)
1440 IF COLPAT(N) = 15 THEN COL(N) = 3: TIL$(N) = CHR$(&H57)
1450 IF COLPAT(N) = 16 THEN COL(N) = 3: TIL$(N) = CHR$(&HAB) + CHR$(&HAB) +
CHR$(&HFF) + CHR$(&HFF)
1460 IF COLPAT(N) = 17 THEN COL(N) = 3: TIL$(N) = CHR$(&H57) + CHR$(&H57) +
CHR$(&HFF) + CHR$(&HFF)
1470 IF COLPAT(N) = 18 THEN COL(N) = 3: TIL$(N) = CHR$(&HFE) + CHR$(&HFA) +
CHR$(&HFA) + CHR$(&HEA) + CHR$(&HFA) + CHR$(&HFE)
1480 IF COLPAT(N) = 19 THEN COL(N) = 3: TIL$(N) = CHR$(&HEB) + CHR$(&HAA) +
CHR$(&HAA) + CHR$(&HEB)
1490 IF COLPAT(N) = 20 THEN COL(N) = 3: TIL$(N) = CHR$(&H77)
1500 IF COLPAT(N) = 21 THEN COL(N) = 3: TIL$(N) = CHR$(&H69) + CHR$(&HAA) +
CHR$(&HAA) + CHR$(&H69)
1510 IF COLPAT(N) = 22 THEN COL(N) = 3: TIL$(N) = CHR$(&HAA) + CHR$(&HBE) +
CHR$(&HBE) + CHR$(&HBE) + CHR$(&HBE) + CHR$(&HAA)
1520 IF COLPAT(N) = 23 THEN COL(N) = 3: TIL$(N) = CHR$(&HE9) + CHR$(&H9E)
1530 IF COLPAT(N) = 24 THEN COL(N) = 3: TIL$(N) = CHR$(&HE9) + CHR$(&HE9)
1540 ' Find maximum and minimum values for x,y,z.

```

(continued)


```

1550 IF X(N) > XMAX THEN XMAX = X(N)
1560 IF X(N) < XMIN THEN XMIN = X(N)
1570 IF Y(N) > YMAX THEN YMAX = Y(N)
1580 IF Y(N) < YMIN THEN YMIN = Y(N)
1590 IF Z(N) > ZMAX THEN ZMAX = Z(N)
1600 IF Z(N) < ZMIN THEN ZMIN = Z(N)
1610 WEND
1620 PRINT N "atoms"
1630 PRINT "rotating..."
1640 ' Find center values for x,y,z.
1650 XCEN = .5*(XMAX+XMIN): YCEN = .5*(YMIN + YMAX): ZCEN = .5*(ZMIN+ZMAX)
1660 ' Rotate molecule around its center.
1670 FOR I = 1 TO N
1680  XA = X(I) - XCEN: YA = Y(I) - YCEN
1690  X(I) = CP*XA+SP*YA: Y(I) = -SP*XA+CP*YA
1700  YA = Y(I): ZA = Z(I) - ZCEN
1710  Y(I) = CT*YA+ST*ZA: Z(I) = -ST*YA+CT*ZA
1715  IF VIEWD < Y(I) THEN CLS: PRINT "Viewing distance is within molecule!
      Rerun with a larger viewing distance.": GOTO 2100
1720 NEXT I: PRINT "sorting..."
1730 '
1740 ' Sort by depth (shell sort).
1750 IGAP = INT(CSNG(N)/2!)
1760 WHILE IGAP >= 1
1770  FOR I = IGAP + 1 TO N
1780   FOR J = I-IGAP TO 1 STEP -IGAP
1790    JG = J + IGAP
1800    IF Y(J) <= Y(JG) THEN GOTO 1850
1810    SWAP X(J),X(JG): SWAP Y(J), Y(JG)
1820    SWAP Z(J), Z(JG): SWAP S(J), S(JG)
1830    SWAP COL(J), COL(JG): SWAP COLPAT(J), COLPAT(JG): SWAP TIL$(J),TIL$(JG)
1840  NEXT J
1850 NEXT I
1860 IGAP = INT(CSNG(IGAP)/2!)
1870 WEND
1880 '
1890 CLS
1900 ' Perspective projection and scale coordinates.
1910 SCALE = -1000000!: SMAX = SCALE
1920 FOR I = 1 TO N
1930  YA = 1!/(VIEWD - Y(I)): X(I) = X(I) *YA: Z(I) = Z(I) * YA: S(I) = S(I)*YA
1940  IF SCALE < ABS(X(I)) THEN SCALE = ABS(X(I))
1950  IF SCALE < ABS(Z(I)) THEN SCALE = ABS(Z(I))
1960  IF SMAX < S(I) THEN SMAX = S(I)
1970 NEXT I: SCALE = 75!/(SCALE + .5*SMAX*SMAG)
1980 SCALEX = SCALE*DISTORT
1990 '
2000 FOR I = 1 TO N
2010  ' Find screen x (ix) and screen z (iz) and screen radius (ir).
2020  ' Center of screen is x = 160 and z = 100.
2030  IX = INT(X(I)*SCALEX + 160!): IZ = INT(Z(I)*SCALE + 100!)
2040  IR = INT(S(I)*SCALE*SMAG): IRZ = IR/DISTORT
2050  COL = COL(I): COLPAT = COLPAT(I): TIL$ = TIL$(I)
2060  GOSUB 2130
2070 NEXT I
2080 CLOSE#1
2090 IF INKEY$ = "" THEN 2090
2100 END
2110 ' Draw patterned circles at ix,iz with radius ir.
2120 ' Draw a circle in color.
2130 CIRCLE (IX,IZ),IR+1,COL
2140 ' Paint the circle black. Start in center and at four extremities
2150 ' in an attempt to completely blacken the circle.
2160 PAINT (IX,IZ),0,COL: PAINT (IX-IR+1,IZ),0,COL: PAINT (IX+IR-1,IZ),0,COL:
PAINT (IX,IZ-IRZ+1),0,COL: PAINT (IX,IZ+IRZ-1),0,COL
2170 ' Paint the circle in color.
2180 PAINT (IX,IZ),COL,COL: PAINT (IX-IR+1,IZ),COL,COL: PAINT (IX+IR
-1,IZ),COL,COL: PAINT (IX,IZ-IRZ+1),COL,COL: PAINT (IX,IZ+IRZ-1),COL,COL
2190 ' Draw circle with a new border color and paint black.
2200 IF COL = 1 THEN COLBOR = 3
2210 IF COL = 2 THEN COLBOR = 3
2220 IF COL = 3 THEN COLBOR = 1
2230 CIRCLE (IX,IZ),IR+1,COLBOR
2240 PAINT (IX,IZ),0,COLBOR
2250 ' Paint circle with final pattern.
2260 IF COLPAT <=3 THEN PAINT (IX,IZ),COL,COLBOR ELSE PAINT

```



```

(IX,IZ),TIL$,COLBOR
2270 ' Draw the circle in black and paint it black.
2280 CIRCLE (IX,IZ),IR+1,0
2290 PAINT (IX,IZ),0,0: PAINT (IX-IR+1,IZ),0,0: PAINT (IX+IR-1,IZ),0,0: PAINT
(IX,IZ-IRZ+1),0,0: PAINT (IX,IZ+IRZ-1),0,0
2300 ' Draw the circle in color and paint with final pattern.
2310 CIRCLE (IX,IZ),IR+1,COLBOR
2320 IF COLPAT <=3 THEN PAINT (IX,IZ),COL,COLBOR ELSE PAINT
(IX,IZ),TIL$,COLBOR
2330 ' Draw the circle in black.
2340 CIRCLE (IX,IZ),IR+1,0
2350 RETURN

```

b5h9.dat

TEXT

Programming Insight: "Molecules in Color," John J. Farrell.
February, page 149. This data file must be used with color3d.bas

5,0,0,1.08676,.83	3,-2.34848,0,.49496,.37
5,3.58,3.58,3.77676,.83	3,1.23152,3.58,3.18496,.37
3,0,0,2.29726,.37	3,0,2.34848,.49496,.37
3,3.58,3.58,4.98726,.37	3,3.58,5.92848,3.18496,.37
5,1.253,0,0,.83	3,0,-2.34848,.49496,.37
5,4.833,3.58,2.69,.83	3,3.58,1.23152,3.18496,.37
5,-1.253,0,0,.83	3,.97376,.97376,-.8877001,.37
5,2.327,3.58,2.69,.83	3,4.55376,4.55376,1.8023,.37
5,0,1.253,0,.83	3,.97376,-.97376,-.8877001,.37
5,3.58,4.833,2.69,.83	3,4.55376,2.60624,1.8023,.37
5,0,-1.253,0,.83	3,-.97376,.97376,-.8877001,.37
5,3.58,2.327,2.69,.83	3,2.60624,4.55376,1.8023,.37
3,2.34848,0,.49496,.37	3,-.97376,-.97376,-.8877001,.37
3,5.92848,3.58,3.18496,.37	3,2.60624,2.60624,1.8023,.37

b10h14.dat

5,.48858,6.9234,0,.88	3,2.80215,6.12616,-1.2518,.38
5,1.437,5.72754,.95592,.88	3,2.48601,3.44072,0,.38
5,1.66692,5.83244,-.74539,.88	3,.44547,3.65052,2.20772,.38
5,1.39389,4.23796,0,.88	3,.70413,6.69262,-1.35991,.38
5,.25866,4.42678,1.35991,.88	3,1.22145,4.42678,-1.21766,.38
5,-.48858,3.5666,0,.88	3,-.58917,2.34976,0,.38
5,-1.437,4.76246,.95592,.88	3,-2.35668,4.69952,1.8208,.38
5,-1.66692,4.65756,-.74539,.88	3,-2.80215,4.36384,-1.2518,.38
5,-1.39389,6.25204,0,.88	3,-2.48601,7.04928,0,.38
5,-.25866,6.06322,1.35991,.88	3,-.44547,6.83948,2.20772,.38
3,.58917,8.14024,0,.38	3,-.70413,3.79738,-1.35991,.38
3,2.35668,5.79048,1.8208,.38	3,-1.22145,6.06322,-1.21766,.38

patterns.dat

TEXT

Programming Insight: "Molecules in Color," John J. Farrell.
February, page 149. This data file must be used with color3d.bas

1,0,0,0,1	9,6,0,3,1
2,3,0,0,1	10,9,0,3,1
3,6,0,0,1	11,12,0,3,1
4,9,0,0,1	12,15,0,3,1
5,12,0,0,1	13,0,0,6,1
6,15,0,0,1	14,3,0,6,1
7,0,0,3,1	15,6,0,6,1
8,3,0,3,1	

(continued)

16,9,0,6,1
17,12,0,6,1
18,15,0,6,1
19,0,0,9,1
20,3,0,9,1

21,6,0,9,1
22,9,0,9,1
23,12,0,9,1
24,15,0,9,1

c7h702n.dat

TEXT

Programming Insight: "Molecules in Color," John J. Farrell.
February, page 149. This data file must be used with color3d.bas

4,6.612408,1.960494,-3.099765,.778
1,2.942464,1.054938,-1.322889,.699
1,4.150042,.732628,-.6921442,.699
1,5.349948,1.056868,-1.255907,.699
1,5.402778,1.690294,-2.49879,.699
1,4.201226,2.029588,-3.136977,.699
1,2.998429,1.706892,-2.556469,.699
1,1.676018,.648866,-.7330775,.699
2,1.585782,.034354,.3349085,.514

2,.5988608,.9981959,-1.402885,.64
3,-.1059083,.5211,-.9861195,.378
3,4.142753,.2509,.2046663,.378
3,6.172514,.91868,-.8186653,.378
3,7.395005,1.86824,-2.586238,.378
3,6.647874,2.61708,-3.7026,.378
3,4.238428,2.57076,-4.037508,.378
3,2.154166,1.8914,-3.032783,.378

cr(c6h6).dat

TEXT

Programming Insight: "Molecules in Color," John J. Farrell.
February, page 149. This data file must be used with color3d.bas

8,0,0,0,1.5
8,0,4.835,4.835,1.5
8,4.835,0,4.835,1.5
8,4.835,4.835,0,1.5
1,1.413754,1.599418,-.1934,.7
1,-1.413754,-1.599418,.1934,.7
1,6.248754,3.235582,.1934,.7
1,3.421246,6.434418,-.1934,.7
1,4.6416,3.421246,-1.599418,.7
1,5.0284,6.248754,1.599418,.7
1,6.434418,5.0284,-1.413754,.7
1,3.235582,4.6416,1.413754,.7
1,1.599418,-.1934,1.413754,.7
1,-1.599418,.1934,-1.413754,.7
1,3.235582,.1934,6.248754,.7
1,6.434418,-.1934,3.421246,.7
1,3.421246,-1.599418,4.6416,.7
1,6.248754,1.599418,5.0284,.7
1,5.0284,-1.413754,6.434418,.7
1,4.6416,1.413754,3.235582,.7
1,-.1934,1.413754,1.599418,.7
1,.1934,-1.413754,-1.599418,.7
1,.1934,6.248754,3.235582,.7
1,-.1934,3.421246,6.434418,.7
1,-1.599418,4.6416,3.421246,.7
1,1.599418,5.0284,6.248754,.7
1,-1.413754,6.434418,5.0284,.7
1,1.413754,3.235582,4.6416,.7
1,.476731,.23208,2.065512,.7
1,-.476731,-.23208,-2.065512,.7
1,5.311732,4.60292,-2.065512,.7
1,4.358269,5.06708,2.065512,.7
1,6.900512,4.358269,-.23208,.7
1,2.769488,5.311732,.23208,.7
1,5.06708,2.769488,-.476731,.7
1,4.60292,6.900512,.476731,.7
1,.23208,2.065512,.476731,.7
1,-.23208,-2.065512,-.476731,.7
1,4.60292,-2.065512,5.311732,.7
1,5.06708,2.065512,4.358269,.7
1,4.358269,-.23208,6.900512,.7
1,5.311732,.23208,2.769488,.7

1,2.769488,-.476731,5.06708,.7
1,6.900512,.476731,4.60292,.7
1,2.065512,.476731,.23208,.7
1,-2.065512,-.476731,-.23208,.7
1,-2.065512,5.311732,4.60292,.7
1,2.065512,4.358269,5.06708,.7
1,-.23208,6.900512,4.358269,.7
1,.23208,2.769488,5.311732,.7
1,-.476731,5.06708,2.769488,.7
1,.476731,4.60292,6.900512,.7
3,1.63423,2.2241,-1.10238,.37
3,-1.63423,-2.2241,1.10238,.37
3,6.46923,2.6109,1.10238,.37
3,3.20077,7.0591,-1.10238,.37
3,3.73262,3.20077,-2.2241,.37
3,5.937381,6.46923,2.2241,.37
3,7.0591,5.937381,-1.63423,.37
3,2.6109,3.73262,1.63423,.37
3,2.2241,-1.10238,1.63423,.37
3,-2.2241,1.10238,-1.63423,.37
3,2.6109,1.10238,6.46923,.37
3,7.0591,-1.10238,3.20077,.37
3,3.20077,-2.2241,3.73262,.37
3,6.46923,2.2241,5.937381,.37
3,5.937381,-1.63423,7.0591,.37
3,3.73262,1.63423,2.6109,.37
3,-1.10238,1.63423,2.2241,.37
3,1.10238,-1.63423,-2.2241,.37
3,1.10238,6.46923,2.6109,.37
3,-1.10238,3.20077,7.0591,.37
3,-2.2241,3.73262,3.20077,.37
3,2.2241,5.937381,6.46923,.37
3,-1.63423,7.0591,5.937381,.37
3,1.63423,2.6109,3.73262,.37
3,.13538,-.36746,2.92034,.37
3,-.13538,.36746,-2.92034,.37
3,4.97038,5.20246,-2.92034,.37
3,4.69962,4.467541,2.92034,.37
3,7.75534,4.69962,.36746,.37
3,1.91466,4.97038,-.36746,.37
3,4.467541,1.91466,-.13538,.37

3,5.20246,7.75534,.13538,.37
 3,-.36746,2.92034,.13538,.37
 3,.36746,-2.92034,-.13538,.37
 3,5.20246,-2.92034,4.97038,.37
 3,4.467541,2.92034,4.69962,.37
 3,4.69962,.36746,7.75534,.37
 3,4.97038,-.36746,1.91466,.37
 3,1.91466,-.13538,4.467541,.37
 3,7.75534,.13538,5.20246,.37

3,2.92034,.13538,-.36746,.37
 3,-2.92034,-.13538,.36746,.37
 3,-2.92034,4.97038,5.20246,.37
 3,2.92034,4.69962,4.467541,.37
 3,.36746,7.75534,4.69962,.37
 3,-.36746,1.91466,4.97038,.37
 3,-.13538,4.467541,1.91466,.37
 3,.13538,5.20246,7.75534,.37

benzene.dat

TEXT

Programming Insight: "Molecules in Color," John J. Farrell.
 February, page 149. This data file must be used with color3d.bas

1,-.424474,1.340674,-.0364068,.699
 1,.424474,-1.340674,.0364068,.699
 1,-.99591,.444636,.8521888,.699
 1,.99591,-.444636,-.8521888,.699
 1,-.577404,-.894105,.873089,.699
 1,.577404,.894105,-.873089,.699

3,-.728096,2.36527,-.1193334,.378
 3,.728096,-2.36527,.1193334,.378
 3,-1.797114,.7674805,1.495376,.378
 3,1.797114,-.7674805,-1.495376,.378
 3,-1.022766,-1.576525,1.55875,.378
 3,1.022766,1.576525,-1.55875,.378

crdibenz.dat

TEXT

Programming Insight: "Molecules in Color," John J. Farrell.
 February, page 149. This data file must be used with color3d.bas

8,0,0,0,1.5
 1,1.413754,1.599418,-.1934,.7
 1,-1.413754,-1.599418,.1934,.7
 1,1.599418,-.1934,1.413754,.7
 1,-1.599418,.1934,-1.413754,.7
 1,-.1934,1.413754,1.599418,.7
 1,.1934,-1.413754,-1.599418,.7
 1,.476731,.23208,2.065512,.7
 1,-.476731,-.23208,-2.065512,.7
 1,.23208,2.065512,.476731,.7
 1,-.23208,-2.065512,-.476731,.7
 1,2.065512,.476731,.23208,.7
 1,-2.065512,-.476731,-.23208,.7

3,1.63423,2.2241,-1.10238,.37
 3,-1.63423,-2.2241,1.10238,.37
 3,2.2241,-1.10238,1.63423,.37
 3,-2.2241,1.10238,-1.63423,.37
 3,-1.10238,1.63423,2.2241,.37
 3,1.10238,-1.63423,-2.2241,.37
 3,.13538,-.36746,2.92034,.37
 3,-.13538,.36746,-2.92034,.37
 3,-.36746,2.92034,.13538,.37
 3,.36746,-2.92034,-.13538,.37
 3,2.92034,.13538,-.36746,.37
 3,-2.92034,-.13538,.36746,.37

nacl.dat

TEXT

Programming Insight: "Molecules in Color," John J. Farrell.
 February, page 149. This data file must be used with color3d.bas

7,0,0,0,.95
 7,5.628,0,0,.95
 7,0,0,5.628,.95
 7,5.628,0,5.628,.95
 7,5.628,2.814,2.814,.95
 7,2.814,2.814,5.628,.95
 7,2.814,2.814,0,.95
 7,2.814,0,2.814,.95
 7,0,2.814,2.814,.95
 7,0,5.628,0,.95
 7,5.628,5.628,0,.95
 7,0,5.628,5.628,.95
 7,5.628,5.628,5.628,.95
 7,2.814,5.628,2.814,.95

6,2.814,2.814,2.814,1.81
 6,2.814,0,0,1.81
 6,0,2.814,0,1.81
 6,0,0,2.814,1.81
 6,5.628,0,2.814,1.81
 6,2.814,0,5.628,1.81
 6,0,2.814,5.628,1.81
 6,5.628,2.814,5.628,1.81
 6,5.628,2.814,0,1.81
 6,0,5.628,2.814,1.81
 6,2.814,5.628,0,1.81
 6,2.814,5.628,5.628,1.81
 6,5.628,5.628,2.814,1.81

(continued)

datagen.bas

TEXT

Programming Insight: "Molecules in Color," John J. Farrell.
February, page 149.

```

100 ' Program to generate a data
    ' file for Cr(C6H6)(CO)3.
105 ' Page 5 of Vol 6 of Crystal
    ' Structures by Wyckoff.
107 ' Unit cell is monoclinic.
110 INPUT "Output file name:";
    FILE$
120 OPEN FILE$ FOR OUTPUT AS #1
130 SIZ=1.4 : COL = 8
140 A = 6.17 : B = 11.07 : C = 6.57
    : BETA = 101.5
150 X = .3319 : Y=.25 : Z = .0225
160 GOSUB 1000
200 SIZ= .7 : COL = 1
    'ring carbons
210 X = .1804 : Y=.3119 : Z =-.2973
220 GOSUB 2000
230 X = .3761 : Y=.3769 : Z =-.2273
240 GOSUB 2000
250 X = .5738 : Y=.3142 : Z =-.1598
260 GOSUB 2000
270 SIZ= .64 'carbonyl carbons
280 X = .5538 : Y=.25 : Z =+.2557
290 GOSUB 1000
300 X = .1827 : Y=.3642 : Z =+.1453
310 GOSUB 2000
320 SIZ= .49: COL = 2
    'carbonyl oxygens
330 X = .6899 : Y=.25 : Z =+.4802
340 GOSUB 1000
350 X = .0894 : Y=.4341 : Z =+.2248
360 GOSUB 2000
400 SIZ= .38: COL = 3 'hydrogens
410 X = .028 : Y=.361 : Z =-.135
420 GOSUB 2000
430 X = .376 : Y=.474 : Z =-.227
440 GOSUB 2000
450 X = .728 : Y=.363 : Z =-.187
460 GOSUB 2000
999 GOTO 5000
1000 WRITE #1, COL,
    (X - Z*SIN
        ((BETA - 90)*3.14159/180))
        *A,Y*B,(Z*COS((BETA -
            90)*3.14159/180))*C,SIZ
1020 RETURN
2000 WRITE #1, COL, (X - Z*SIN
    ((BETA - 90)*3.14159/180))
    *A,Y*B,(Z*COS((BETA -
        90)*3.14159/180))*C,SIZ
2020 WRITE #1, COL, (X - Z*SIN
    ((BETA - 90)*3.14159/180))
    *A, (.5-Y)*B,(Z*COS((BETA -
        90)*3.14159/180))*C,SIZ
2040 RETURN
5000 CLOSE #1: END

```

dvorak.txt

TEXT

"Keyboard Efficiency." See dvorak.bas for details.

Text is entered line-by-line. A line is entered by the RETURN key or is automatically accepted if the cursor reaches column 39; warning beeps are sounded past column 31. After a line is entered, it is retyped by the computer as it is analyzed. Editing is possible within a line by backing up with the left-arrow and retyping. Once a line is analyzed it cannot be edited further. We suggest typing with full use of upper and lower case. However, the program will run correctly if the CAPS LOCK is locked down for the entire file; in that case, it will infer use of the shift key for shifted special symbols, but not for capital letters.

TO UNDERSTAND THE LISTING:

The fingers (excluding the thumbs) are numbered 1 to 8 from left to right. The label K refers to Qwerty (K = 1) or Dvorak (K = 2). The variables CX(F,K) and CY(F,K) are the current column and row locations of finger F in keyboard K, relative to X = Y = 0 (the home position of that finger). Total motion FT(F,K) is incremented by an amount from a look-up table of distances, in actual inches, called DS(I,J). The DS(I,J) are computed by approximating the keyboard as a grouping of parallelograms with sides 0.75" (horizontal key spacing) and 0.81" (measured along a diagonal from the upper left to the lower right, tilted approximately 23 degrees from the vertical). Like the Qwerty layout itself, this parallelogram pattern dates from the early days of mechanical typewriters when it was necessary to leave room for the connecting levers beneath the keys. A sample DATA line (e.g., 1066 DATA 66,B,4,1,-1,8,5,-1,-1,1) is interpreted as follows. ASCII code 66 is a capital B. In Qwerty, it is typed by finger 4, moved one column to the

right and one row down relative to the home position; also, if CAPS LOCK is off, finger 8 (the little finger on the right hand) must reach down for the SHIFT key. In the Dvorak layout, B is typed by finger 5, one column to the left and one row down from the home position; if CAPS LOCK is off, finger 1 must strike the SHIFT key. Characters with ASCII codes above 122 or below 32 (mainly control characters) are ignored, except for left-arrow (ASCII 8), RETURN (ASCII 13), and ESCAPE (ASCII 27).

simp13.bix

TEXT

Programming Project: "A Simpl Compiler Part 3: Extensions," by Jonathan Amsterdam.
February, page 102. Also download reads13.me.

=====

Break the following modules out into their own files
Start CodeGen.DEF

DEFINITION MODULE CodeGen;

(* This module generates code from parse trees. *)

FROM Node IMPORT node;
FROM Symbol IMPORT symbol;

EXPORT QUALIFIED genBlock, genGlobal, genLocals;

PROCEDURE genBlock(n:node);
(* Generate code for a block of statements. *)

PROCEDURE genGlobal(s:symbol);
(* Generate code for a global variable. *)

PROCEDURE genLocals(routine:symbol);
(* Generate code to set up the stack for local variables. *)

END CodeGen.

=====

Start CodeGen.MOD

IMPLEMENTATION MODULE CodeGen;

(* Code Generator for the SIMPL compiler.

Changes for part 3:

1. Arrays handled. No arrays are initialized. Now, also, locals of any type are not initialized.
2. IN/OUT handled.
3. Array and string const assignment handled.
4. String constant assignment handled.

*)

FROM MyTerminal IMPORT fatal;
IMPORT MyTerminal;
FROM InOut IMPORT WriteString, WriteLn, WriteCard;
FROM Node IMPORT node, nodeClass, NodeClass, nodeEmpty, nodeFirst, nodeRest, nodeTest, nodeThen, nodeElse, nodeStmts, nodeRHS, nodeLHS, nodeArgs, nodeRoutine, nodeExpr, nodeArg, nodeLeftArg, nodeRightArg, nodeOp, nodeSymbol, nodeInt, nodeBool, nodeNumFormals, nodeChar, nodeType, freeNode, nodeArray, nodeIndex, nodeString;
FROM CodeWrite IMPORT writeLabel, writeStringLabel, writeStringBranch, writeCondBranch, writeBranch, writeSymPop, writeCall, writeChar, writeWriteInt, writeInt, writeReadInt, writeReturn, writeFReturn, writeOp, writeBool, writeSymbol, writeWriteChar, writeReadChar, writeSetSP, writePop, writeLow, writeHigh, writeCopy, writeAddr, writeMin, writeContents, writeAref, recordString;
FROM Token IMPORT tokenClass, isRelation, stringType;
FROM LexAn IMPORT errorFlag;
FROM Symbol IMPORT symbol, numLocals, Class, modeType, symbolList, s1Symbol, s1Next, s1Empty;

(continued)

```

IMPORT Symbol;
FROM SymbolTable IMPORT tString, lowFunc, highFunc, tInteger, tChar, tBoolean;
FROM StringStuff IMPORT stringLen;
FROM TypeChecker IMPORT baseType;

```

```

(** label generation **)

```

```

(* The code generator needs a supply of unique label names. *)

```

```

MODULE LabelGenerator;
EXPORT newLabel, label;

```

```

    TYPE label = CARDINAL;

```

```

    VAR labelCount: CARDINAL;

```

```

    PROCEDURE newLabel(): label;

```

```

    BEGIN

```

```

        INC(labelCount);

```

```

        RETURN label(labelCount);

```

```

    END newLabel;

```

```

BEGIN

```

```

    labelCount := 0;

```

```

END LabelGenerator;

```

```

PROCEDURE genBlock(n: node);

```

```

(* This is the interface to generating statements. We don't waste our time
   doing generation if there has been an error. Also, it's possible for this
   routine to get an empty node (legally); in that case, we do nothing. *)

```

```

BEGIN

```

```

    IF (NOT errorFlag) AND (NOT nodeEmpty(n)) THEN

```

```

        IF nodeClass(n) <> nList THEN

```

```

            MyTerminal.fatal('genBlock: node class must be nList');

```

```

        ELSE

```

```

            genStmts(n);

```

```

            freeNode(n);

```

```

        END;

```

```

    END;

```

```

END genBlock;

```

```

PROCEDURE genGlobal(s: symbol);

```

```

(* Output the global symbol as a label. Initialize integers to 0, booleans
   to FALSE (which is also zero), chars to NUL (which is again zero).
   For arrays, just write a .BLOCK. No initialization. *)

```

```

VAR name: stringType;

```

```

    size: CARDINAL;

```

```

BEGIN

```

```

    IF NOT errorFlag THEN

```

```

        IF Symbol.class(s) = Global THEN

```

```

            Symbol.string(s, name);

```

```

            size := Symbol.size(Symbol.type(s));

```

```

            writeStringLabel(name);

```

```

            IF size = 1 THEN

```

```

                WriteString(" 0");

```

```

            ELSE

```

```

                WriteString(" .BLOCK ");

```

```

                WriteCard(size, 0);

```

```

            END;

```

```

            WriteLn;

```

```

        ELSE

```

```

            MyTerminal.WriteString("genGlobal: not a global: ");

```

```

            fatal(name);

```

```

        END;

```

```

    END;

```

```

END genGlobal;

```

```

PROCEDURE genLocals(routine: symbol);

```

```

(* save space for locals on stack *)

```

```

VAR n: CARDINAL;

```

```

BEGIN

```

```

    n := sizeLocals(routine);

```

```

    IF n <> 0 THEN

```

```

        writeSetSP(INTEGER(n));

```

```

    END;

```



```
END genLocals;
```

```
PROCEDURE sizeLocals(routine:symbol):CARDINAL;
VAR locals:symbolList;
    sum:CARDINAL;
BEGIN
    locals := Symbol.locals(routine);
    sum := 0;
    WHILE NOT sIsEmpty(locals) DO
        INC(sum, Symbol.size(Symbol.type(s1Symbol(locals))));
        locals := s1Next(locals);
    END;
    RETURN sum;
END sizeLocals;
```

```
PROCEDURE genStmts(n:node);
(* Generate a list of statements, if the node isn't empty. *)
BEGIN
    IF NOT nodeEmpty(n) THEN
        genStmt(nodeFirst(n));
        genStmts(nodeRest(n));
    END;
END genStmts;
```

(*** Statements ***)

```
PROCEDURE genStmt(n:node);
BEGIN
    CASE nodeClass(n) OF
        nIf: genIfStmt(n);
        nWhile: genWhileStmt(n);
        nAssignment: genAssignStmt(n);
        nCall: genCallStmt(n);
        nWrite: genWriteStmt(n);
        nRead: genReadStmt(n);
        nReturn: genReturnStmt(n);
    ELSE
        MyTerminal.fatal("genStmt: unknown statement type");
    END;
END genStmt;
```

```
PROCEDURE genIfStmt(n:node);
VAR label1, label2:label;
BEGIN
    label1 := newLabel();
    genExpr(nodeTest(n)); (* generate test *)
    writeCondBranch(Equal, label1); (* branch to else part if test false *)
    genBlock(nodeThen(n)); (* generate then part *)
    IF nodeEmpty(nodeElse(n)) THEN (* no else part *)
        writeLabel(label1);
    ELSE
        label2 := newLabel();
        writeBranch(label2); (* branch around els part *)
        writeLabel(label1); (* label for else part *)
        genBlock(nodeElse(n)); (* generate else part *)
        writeLabel(label2); (* final label *)
    END;
END genIfStmt;
```

```
PROCEDURE genWhileStmt(n:node);
VAR testLabel, endLabel:label;
BEGIN
    testLabel := newLabel();
    endLabel := newLabel();
    writeLabel(testLabel); (* label for top of loop *)
    genExpr(nodeTest(n)); (* generate test *)
    writeCondBranch(Equal, endLabel); (* if false, branch to end of loop *)
    genBlock(nodeStmts(n)); (* generate loop body *)
    writeBranch(testLabel); (* branch back to test *)
    writeLabel(endLabel); (* end label *)
END genWhileStmt;
```

```
PROCEDURE genAssignStmt(n:node);
BEGIN
```

(continued)

```

IF Symbol.class(baseType(nodeType(nodeLHS(n)))) = ArrayType THEN
  (* RHS had better be an array type too, or a string const *)
  genIndex(nodeRHS(n));
  genIndex(nodeLHS(n));
  computeSize(nodeRHS(n), nodeLHS(n));
  writeCopy;
ELSE
  genExpr(nodeRHS(n));          (* generate the expression *)
  genScalarAssign(nodeLHS(n));
END;
END genAssignStmt;

PROCEDURE genScalarAssign(n:node);
(* generate code to assign top of stack to n *)
BEGIN
  IF nodeClass(n) = nIndex THEN
    genIndex(n);
    writePop;
  ELSIF Symbol.class(baseType(nodeType(n))) = ArrayType THEN (* error *)
    fatal('genScalarAssign: array type given');
  ELSE (* a scalar variable *)
    writeSymPop(nodeSymbol(n)); (* pop result into the variable *)
  END;
END genScalarAssign;

PROCEDURE computeSize(source, dest:node);
(* Size for an array copy is tricky. The rules are:
  1. If the size of source and dest are known (i.e. they are not open
     arrays), then take the min.
  2. If the size of one or both isn't known, compute the min at runtime. *)
VAR stype, dtype:symbol;
BEGIN
  stype := baseType(nodeType(source));
  dtype := baseType(nodeType(dest));
  IF Symbol.open(stype) OR Symbol.open(dtype) THEN
    genSize(source, stype);
    genSize(dest, dtype);
    writeMin; (* MIN instruction added for this purpose *)
  ELSIF nodeClass(source) = nString THEN
    IF nstringLen(source) + 1 < Symbol.size(dtype) THEN
      genSize(source, stype);
    ELSE
      genSize(dest, dtype);
    END;
  ELSIF NOT Symbol.equal(stype, dtype) THEN
    fatal('computeSize: not same type');
  ELSE
    genSize(source, stype);
  END;
END computeSize;

PROCEDURE genSize(n:node; ntype:symbol);
(* generate code to put size of n on stack; n must be of type array. *)
BEGIN
  IF Symbol.open(ntype) THEN (* size = highBound - lowBound + 1 *)
    genHighBound(n);
    genLowBound(n);
    writeOp(Minus);
    writeInt(1);
    writeOp(Plus);
  ELSIF nodeClass(n) = nString THEN (* string const *)
    writeInt(INTEGER(nstringLen(n) + 1));
    (* Important: +1 for null at end *)
  ELSE (* a non-open array variable *)
    writeInt(INTEGER(Symbol.size(ntype)));
  END;
END genSize;

PROCEDURE genCallStmt(n:node);
BEGIN
  (* Here we do the special checks for the pseudofunctions LOW and HIGH *)
  IF Symbol.equal(nodeRoutine(n), lowFunc) THEN
    genLowBound(nodeFirst(nodeArgs(n)));
  ELSIF Symbol.equal(nodeRoutine(n), highFunc) THEN
    genHighBound(nodeFirst(nodeArgs(n)));
  ELSE (* a "real" function *)

```



```

        genArgs(nodeRoutine(n), nodeArgs(n)); (* generate the arguments *)
        writeCall(nodeRoutine(n));          (* generate a call instruction *)
    END;
END genCallStmt;

PROCEDURE genArgs(routine:symbol; arglist:node);
(* Iterate down the arglist and the list of the routine's formals,
   and generate code for each argument *)
VAR formlist:symbolList;
BEGIN
    formlist := Symbol.formals(routine);
    (* assume number of formals = number of args *)
    WHILE NOT nodeEmpty(arglist) DO
        genArg(nodeFirst(arglist), slSymbol(formlist));
        arglist := nodeRest(arglist);
        formlist := slNext(formlist);
    END;
END genArgs;

PROCEDURE genArg(arg:node; formal:symbol);
(* Generate code to push the argument on the stack, as follows:
   1. Argument is an array:
       1a. If formal is open array, push high bound, low bound;
       1b. Push starting address, regardless
   2. Argument is a scalar:
       2a. If formal has mode IN, do the usual thing: genExpr.
       2b. Otherwise, push the address of the scalar.
   *)
VAR argType:symbol;
BEGIN
    argType := baseType(nodeType(arg));
    IF Symbol.class(argType) = ArrayType THEN
        IF Symbol.open(Symbol.type(formal)) THEN
            genHighBound(arg);
            genLowBound(arg);
        END;
        genIndex(arg);
    ELSIF Symbol.class(argType) = ScalarType THEN
        IF Symbol.mode(formal) = min THEN
            genExpr(arg);
        ELSE
            genIndex(arg);
        END;
    ELSE
        fatal('genArg: argtype not a type object');
    END;
END genArg;

PROCEDURE genWriteStmt(n:node);
(* Generate code to write the arguments to the screen. WRITE can take any
   number of arguments. *)
VAR arglist:node;
    argType:symbol;
BEGIN
    arglist := nodeArgs(n);
    WHILE NOT nodeEmpty(arglist) DO
        genExpr(nodeFirst(arglist));
        argType := baseType(nodeType(nodeFirst(arglist)));
        IF Symbol.equal(argType, tInteger) THEN
            writeWriteInt;
        ELSE (* it's a char *)
            writeWriteChar;
        END;
        arglist := nodeRest(arglist);
    END;
END genWriteStmt;

PROCEDURE genReadStmt(n:node);
(* Generate code to read from the terminal. READ can take any number of
   arguments. *)
VAR arglist:node;
    argType:symbol;
BEGIN
    arglist := nodeArgs(n);
    WHILE NOT nodeEmpty(arglist) DO

```

(continued)

```

    argType := baseType(nodeType(nodeFirst(arglist)));
    IF Symbol.equal(argType, tInteger) THEN
        writeReadInt;
        genScalarAssign(nodeFirst(arglist));
    ELSE (* a char *)
        writeReadChar;
        genScalarAssign(nodeFirst(arglist));
    END;
    arglist := nodeRest(arglist);
END;
END genReadStmt;

PROCEDURE genReturnStmt(n:node);
BEGIN
    IF nodeEmpty(nodeExpr(n)) THEN (* a procedure return *)
        writeReturn(nodeNumFormals(n));
    ELSE (* a function return *)
        genExpr(nodeExpr(n));
        writeFReturn(nodeNumFormals(n));
    END;
END genReturnStmt;

(** expressions **)

PROCEDURE genExpr(n:node);
BEGIN
    CASE nodeClass(n) OF
        nUnop: genExpr(nodeArg(n));
                writeOp(nodeOp(n));
        | nOp: IF (nodeOp(n) = And) OR (nodeOp(n) = Or) THEN
                genLogicalOp(n);
            ELSE
                genExpr(nodeLeftArg(n));
                genExpr(nodeRightArg(n));
                writeOp(nodeOp(n));
            END;
        | nInt: writeInt(nodeInt(n));
        | nBool: writeBool(nodeBool(n));
        | nChar: writeChar(nodeChar(n));
        | nSymbol: writeSymbol(nodeSymbol(n));
        | nCall: genCallStmt(n);
        | nIndex: genIndex(n);
                writeContents;
    ELSE
        MyTerminal.fatal("genExpr: unknown expression type");
    END;
END genExpr;

PROCEDURE genLogicalOp(n:node);
(* AND's and OR's end up here. We generate code to evaluate only the first
   if possible. *)
VAR label1, label2:label;
BEGIN
    label1 := newLabel();
    label2 := newLabel();
    genExpr(nodeLeftArg(n));
    IF nodeOp(n) = And THEN (* we branch to FALSE if the first was FALSE *)
        writeCondBranch(Equal, label1);
    ELSE (* it's OR; we branch to TRUE if the first was TRUE *)
        writeCondBranch(Greater, label1);
    END;
    genExpr(nodeRightArg(n)); (* if the first one failed to decide, the value
                               of the 2nd is the value of the whole thing. *)
    writeBranch(label2);
    writeLabel(label1);
    writeBool(nodeOp(n) = Or); (* write TRUE if OR, FALSE if AND *)
    writeLabel(label2);
END genLogicalOp;

PROCEDURE genIndex(n:node);
(* Generates an array index. Ends up with element address on top of stack.
   Works fine for whole arrays, too--puts the starting address on the stack.
   In fact, works fine for scalar symbols too! And for strings. *)
VAR arrayVar:node;
BEGIN
    IF nodeClass(n) = nSymbol THEN
        writeAddr(nodeSymbol(n));
    
```



```

    ELSIF nodeClass(n) = nString THEN
        genString(n);
    ELSIF nodeClass(n) = nIndex THEN
        arrayVar := nodeArray(n);
        genIndex(arrayVar);
        genLowBound(arrayVar);
        genHighBound(arrayVar);
        genExpr(nodeIndex(n));
        writeAref(Symbol.size(baseType(nodeType(n))));
    ELSE
        fatal('genIndex: node not symbol, string or index');
    END;
END genIndex;

PROCEDURE genLowBound(arrayVar:node);
VAR arrayType:symbol;
BEGIN
    arrayType := baseType(nodeType(arrayVar));
    IF nodeClass(arrayVar) = nSymbol THEN
        IF Symbol.open(arrayType) THEN
            writeLow(nodeSymbol(arrayVar));
        ELSE
            writeInt(Symbol.lowBound(arrayType));
        END;
    ELSIF nodeClass(arrayVar) = nString THEN
        writeInt(0);
    ELSIF nodeClass(arrayVar) = nIndex THEN
        writeInt(Symbol.lowBound(arrayType));
    ELSE
        fatal('genLowBound: not symbol, string or index');
    END;
END genLowBound;

PROCEDURE genHighBound(arrayVar:node);
VAR arrayType:symbol;
BEGIN
    arrayType := baseType(nodeType(arrayVar));
    IF nodeClass(arrayVar) = nSymbol THEN
        IF Symbol.open(arrayType) THEN
            writeHigh(nodeSymbol(arrayVar));
        ELSE
            writeInt(Symbol.highBound(arrayType));
        END;
    ELSIF nodeClass(arrayVar) = nString THEN
        writeInt(nstringLen(arrayVar)-1); (* does not include 0 at end *)
    ELSIF nodeClass(arrayVar) = nIndex THEN
        writeInt(Symbol.highBound(arrayType));
    ELSE
        fatal('genHighBound: not symbol or index');
    END;
END genHighBound;

PROCEDURE genString(n:node);
VAR lab:label;
    s:stringType;
BEGIN
    lab := newLabel();
    nodeString(n, s);
    recordString(lab, s);
END genString;

PROCEDURE nstringLen(n:node):CARDINAL;
VAR s:stringType;
BEGIN
    nodeString(n, s);
    RETURN stringLen(s);
END nstringLen;

BEGIN
END CodeGen.

```

```

=====
Start CodeWrite.DEF
=====

```

(continued)

```
DEFINITION MODULE CodeWrite;
```

```
(* This module outputs the code for the SIMPL compiler. *) \
```

```
FROM Symbol IMPORT symbol;
```

```
FROM Token IMPORT tokenClass, stringType;
```

```
EXPORT QUALIFIED writeLabel, writeStringLabel, writeRoutineLabel, writeHalt,
  writeStringBranch, writeBranch, writeCondBranch, writePop, writeCall,
  writeWriteInt, writeReadInt, writeReturn, writeFReturn, writeOp,
  writeInt, writeBool, writeSymbol, writeChar, writeSymPop,
  writeWriteChar, writeReadChar, writeLow, writeHigh, writeAddr,
  writeCopy, writeMin, writeContents, writeSetSP, writeAref,
  recordString, writeStrings;
```

```
PROCEDURE writeLabel(c:CARDINAL);
```

```
(* Writes an "L" followed by the number, then a colon. *)
```

```
PROCEDURE writeStringLabel(s:ARRAY OF CHAR);
```

```
(* Just writes the string followed by a colon. *)
```

```
PROCEDURE writeRoutineLabel(routine:symbol);
```

```
(* Writes the name of the routine followed by a colon, and writes (on the
  screen) the procedure name so the user knows it's being compiled. *)
```

```
PROCEDURE writeStringBranch(s: ARRAY OF CHAR);
```

```
(* Write a branch followed by the string *)
```

```
PROCEDURE writeCondBranch(tc:tokenClass; c:CARDINAL);
```

```
(* Write a conditional branch (Equal, Greater or Less) followed by "L", then
  the number. *)
```

```
PROCEDURE writeBranch(c:CARDINAL);
```

```
(* Write an unconditional branch to the label. *)
```

```
PROCEDURE writeCall(s:symbol);
```

```
(* Generate a call instruction with the symbol *)
```

```
PROCEDURE writeWriteInt;
```

```
PROCEDURE writeReadInt;
```

```
PROCEDURE writeWriteChar;
```

```
PROCEDURE writeReadChar;
```

```
(* instructions for I/O *)
```

```
PROCEDURE writeReturn(numFormals:CARDINAL);
```

```
PROCEDURE writeFReturn(numFormals:CARDINAL);
```

```
(* Two types of return instructions; both take the number of formals as arg. *)
```

```
PROCEDURE writeOp(tc:tokenClass);
```

```
(* Write the instruction corresponding to the operator *)
```

```
PROCEDURE writeInt(i:INTEGER);
```

```
PROCEDURE writeBool(b:BOOLEAN);
```

```
PROCEDURE writeChar(c:CHAR);
```

```
(* Write pushes for these constants. *)
```

```
PROCEDURE writeSymbol(s:symbol);
```

```
(* Generate the appropriate push instruction for the symbol *)
```

```
PROCEDURE writeSymPop(s:symbol);
```

```
(* Generate the appropriate pop instruction for the symbol *)
```

```
PROCEDURE writeHalt;
```

```
PROCEDURE writeAddr(s:symbol);
```

```
(* Put the address of the symbol on the stack *)
```

```
PROCEDURE writeLow(s:symbol);
```

```
PROCEDURE writeHigh(s:symbol);
```

```
(* For low and high bounds of open arrays. *)
```

```
PROCEDURE writeCopy;
```

```
PROCEDURE writeMin;
```

```
PROCEDURE writeContents;
```

```
PROCEDURE writePop;
```

```
PROCEDURE writeSetSP(i:INTEGER);
```

```
PROCEDURE writeAref(size:CARDINAL);
```



```

PROCEDURE recordString(lab:CARDINAL; VAR s:stringType);
(* remembers a string, to be output later *)

PROCEDURE writeStrings;
(* Writes out remembered strings, preceded by their labels. *)

END CodeWrite.

=====
Start CodeWrite.MOD
=====

IMPLEMENTATION MODULE CodeWrite;

FROM InOut IMPORT WriteString, WriteLn, WriteInt, WriteCard;
(* We can't use Write because of a conflict inside tokenClass *)
IMPORT InOut;
FROM Symbol IMPORT symbol, Class, modeType;
IMPORT Symbol;
FROM SymbolTable IMPORT currentLexLevel;
FROM Token IMPORT tokenClass, stringType;
IMPORT MyTerminal;

PROCEDURE writeStringLabel(s:ARRAY OF CHAR);
BEGIN
    WriteString(s);
    WriteString(': ');
END writeStringLabel;

PROCEDURE writeRoutineLabel(routine:symbol);
VAR name:stringType;
BEGIN
    writeRoutineName(routine);
    WriteString(': ');
    WriteLn;
    Symbol.string(routine, name);
    MyTerminal.WriteString(name);
    MyTerminal.WriteLineString("...");
END writeRoutineLabel;

PROCEDURE writeRoutineName(routine:symbol);
VAR name:stringType;
BEGIN
    Symbol.string(routine, name);
    WriteString(name);
    IF Symbol.lexLevel(routine) <> 0 THEN
        WriteInt(Symbol.offset(routine), 0);
    END;
END writeRoutineName;

PROCEDURE writeLabel(c:CARDINAL);
BEGIN
    InOut.Write('L');
    WriteCard(c, 0);
    InOut.Write(': ');
    WriteLn;
END writeLabel;

PROCEDURE writeStringBranch(s:ARRAY OF CHAR);
BEGIN
    writeOpCode('BRANCH ');
    WriteLineString(s);
END writeStringBranch;

PROCEDURE writeBranch(c:CARDINAL);
BEGIN
    writeOpCode('BRANCH L');
    WriteCard(c, 0);
    WriteLn;
END writeBranch;

PROCEDURE writeCondBranch(tc:tokenClass; c:CARDINAL);
BEGIN
    CASE tc OF
        Equal: writeOpCode('BREQL L');
        | Less: writeOpCode('BRLSS L');
    END;

```

(continued)

```

      | Greater:writeOpCode('BRGTR  L');
    ELSE
      MyTerminal.fatal('writeCondBranch: unknown branch type');
    END;
    WriteCard(c, 0);
    WriteLn;
  END writeCondBranch;

PROCEDURE writeWriteInt;
BEGIN
  writeOpCode('WRINT');
  WriteLn;
END writeWriteInt;

PROCEDURE writeReadInt;
BEGIN
  writeOpCode('RDINT');
  WriteLn;
END writeReadInt;

PROCEDURE writeWriteChar;
BEGIN
  writeOpCode('WRCHAR');
  WriteLn;
END writeWriteChar;

PROCEDURE writeReadChar;
BEGIN
  writeOpCode('RDCHAR');
  WriteLn;
END writeReadChar;

PROCEDURE writeHalt;
BEGIN
  writeOpCode('HALT');
  WriteLn;
END writeHalt;

PROCEDURE writeReturn(numFormals:CARDINAL);
BEGIN
  writeOpCode('RETURN  ');
  WriteCard(numFormals, 0);
  WriteLn;
END writeReturn;

PROCEDURE writeFReturn(numFormals:CARDINAL);
BEGIN
  writeOpCode('FRETURN ');
  WriteCard(numFormals, 0);
  WriteLn;
END writeFReturn;

PROCEDURE writeInt(i:INTEGER);
BEGIN
  writeOpCode('PUSHC  ');
  WriteInt(i, 0);
  WriteLn;
END writeInt;

PROCEDURE writeChar(c:CHAR);
BEGIN
  writeOpCode('PUSHC  ');
  InOut.Write('');
  InOut.Write(c);
  WriteLn;
END writeChar;

PROCEDURE writeBool(b:BOOLEAN);
BEGIN
  IF b THEN
    writeInt(1);
  ELSE
    writeInt(0);
  END;
END writeBool;

```



```

PROCEDURE writeSymPop(s:symbol);
(* Not defined on array symbols. *)
BEGIN
  writeSymAddr(s, "POPC   ", "POPL   ", "PUSHL  ");
  IF Symbol.class(s) = Formal THEN
    writePop;
  END;
END writeSymPop;

PROCEDURE writeSymbol(s:symbol);
(* Put the contents of the symbol on the stack. Not defined for
array symbols. *)
BEGIN
  writeSymAddr(s, "PUSH   ", "PUSHL  ", "PUSHL  ");
  IF (Symbol.class(s) = Formal) AND (Symbol.mode(s) = mInOut) THEN
    writeContents;
  END;
END writeSymbol;

PROCEDURE writeAddr(s:symbol);
(* Put the address of the symbol on the stack. Rules:
array:
  global: pushc;
  local:  addrl;
  formal: pushl;
scalar:
  global: pushc;
  local:  addrl;
  formal: IN: addrl; OUT, IN OUT; pushl
*)
BEGIN
  IF (Symbol.class(Symbol.type(s)) = ScalarType) AND
    (Symbol.class(s) = Formal) AND
    (Symbol.mode(s) = mIn) THEN
    writeSymAddr(s, "dummy", "dummy", "ADDRL  ");
  ELSE
    writeSymAddr(s, "PUSHC   ", "ADDRL  ", "PUSHL  ");
  END;
END writeAddr;

PROCEDURE writeSymAddr(s:symbol; global, local, formal:ARRAY OF CHAR);
VAR name:stringType;
BEGIN
  Symbol.string(s, name);
  IF Symbol.class(s) = Global THEN
    writeOpCode(global);
    WriteLnString(name);
  ELSE
    IF Symbol.class(s) = Local THEN
      writeOpCode(local);
    ELSIF Symbol.class(s) = Formal THEN
      writeOpCode(formal);
    ELSE
      MyTerminal.fatal('writeSymAddr: not variable');
    END;
    WriteInt(currentLexLevel() - Symbol.lexLevel(s), 0);
    WriteString(', ');
    WriteInt(Symbol.offset(s), 0);
    writeComment(name);
  END;
END writeSymAddr;

PROCEDURE writeCall(s:symbol);
BEGIN
  writeOpCode("CALL   ");
  writeRoutineName(s);
  WriteString(", ");
  WriteInt(currentLexLevel() - Symbol.lexLevel(s), 0);
  WriteLn;
END writeCall;

PROCEDURE writeOp(tc:tokenClass);
BEGIN
  CASE tc OF
    Plus:      writeOpCode('ADD');
    Minus:     writeOpCode('SUB');
  
```

(continued)

```

    UMinus:      writeOpCode('NEG');
    Times:       writeOpCode('MUL');
    Divide:      writeOpCode('DIV');
    Not:         writeOpCode('NOT');
    Equal:       writeOpCode('EQUAL');
    Greater:     writeOpCode('GREATER');
    Less:        writeOpCode('LESS');
    NotEqual:    writeOpCode('NOTEQL');
    LessEqual:   writeOpCode('LSSEQL');
    GreaterEqual: writeOpCode('GTREQL');
ELSE MyTerminal.fatal("writeOp: unknown operator");
END;
WriteLn;
END writeOp;

PROCEDURE WriteLnString(s:ARRAY OF CHAR);
BEGIN
    WriteString(s);
    WriteLn;
END WriteLnString;

PROCEDURE writeOpCode(s:ARRAY OF CHAR);
BEGIN
    WriteString(" ");
    WriteString(s);
END writeOpCode;

PROCEDURE writeComment(s:ARRAY OF CHAR);
BEGIN
    WriteString("    ; ");
    WriteString(s);
    WriteLn;
END writeComment;

PROCEDURE writeCopy;
BEGIN
    writeOpCode("COPY");
    WriteLn;
END writeCopy;

PROCEDURE writeMin;
BEGIN
    writeOpCode("MIN");
    WriteLn;
END writeMin;

PROCEDURE writePop;
BEGIN
    writeOpCode("POP");
    WriteLn;
END writePop;

PROCEDURE writeContents;
BEGIN
    writeOpCode("CONTENTS");
    WriteLn;
END writeContents;

PROCEDURE writeLow(s:symbol);
(* stack looks like this:
    | highBound |
    | lowBound  |
    | startaddr | <-- offset
*)
BEGIN
    writeOpCode("PUSHL ");
    WriteInt(currentLexLevel() - Symbol.lexLevel(s), 0);
    WriteString(', ');
    WriteInt(Symbol.offset(s)+1, 0);
    writeComment("LOW");
END writeLow;

PROCEDURE writeHigh(s:symbol);
(* stack looks like this:
    | highBound |

```



```

      | lowBound |
      | startaddr | <-- offset
*)
BEGIN
  writeOpCode("PUSHL  ");
  WriteInt(currentLexLevel() - Symbol.lexLevel(s), 0);
  WriteString(', ');
  WriteInt(Symbol.offset(s)+2, 0);
  writeComment("HIGH");
END writeHigh;

PROCEDURE writeSetSP(i:INTEGER);
BEGIN
  writeOpCode("SETSP  ");
  WriteInt(i, 0);
  WriteLn;
END writeSetSP;

PROCEDURE writeAref(size:CARDINAL);
BEGIN
  writeOpCode("AREF   ");
  WriteCard(size, 0);
  WriteLn;
END writeAref;

MODULE Strings;      (* for handling string constants *)
FROM MyTerminal IMPORT fatal;
FROM InOut IMPORT Write, EOL;
IMPORT writeLabel, WriteString, WriteLn, WriteCard, stringType,
       writeOpCode, writeComment;
EXPORT writeStrings, recordString;

CONST  maxStrings = 20; (* max string consts per routine *)
TYPE stringRec = RECORD
      string:stringType;
      label:CARDINAL;
    END;
VAR strings: ARRAY[1..maxStrings] OF stringRec;
    nStrings:[0..maxStrings];

PROCEDURE recordString(lab:CARDINAL; VAR s:stringType);
(* record the string in the array and write code to push its address *)
BEGIN
  IF nStrings = maxStrings THEN
    fatal('too many strings in routine');
  ELSE
    INC(nStrings);
    WITH strings[nStrings] DO
      label := lab;
      string := s;
    END;
    writeOpCode('PUSHC  L');
    WriteCard(lab, 0);
    WriteLn;
  END;
END recordString;

PROCEDURE writeStrings;
VAR i:CARDINAL;
BEGIN
  FOR i := 1 TO nStrings DO
    writeLabel(strings[i].label);
    WriteString(' ');
    formatString(strings[i].string);
    WriteString(' ');
    WriteCard(0, 0);
    WriteLn;
  END;
  nStrings := 0;
END writeStrings;

PROCEDURE formatString(VAR s:ARRAY OF CHAR);
(* puts slashes back in *)
CONST Tab = 11C;
VAR i:CARDINAL;

```

(continued)

```

BEGIN
  i := 0;
  WHILE (i <= HIGH(s)) AND (s[i] <> 0C) DO
    IF s[i] = ' ' THEN
      WriteString(' ');
    ELSIF s[i] = EOL THEN
      WriteString("\n");
    ELSIF s[i] = Tab THEN
      WriteString("\t");
    ELSIF s[i] = "\" THEN
      WriteString("\\");
    ELSE
      Write(s[i]);
    END;
    INC(i);
  END;
END formatString;

```

```

BEGIN (* module Strings *)
  nStrings := 0;
END Strings;
BEGIN
END CodeWrite.

```

```

=====
Start Compiler.MOD
=====

```

```

MODULE Compiler;

```

```

(* A compiler for the SIMPL programming language.
Copyright 1985 by Jonathan Amsterdam. All Rights Reserved.
See the BYTE article "A SIMPL Compiler" for more information.

```

Module map, roughly in order of low to high level:

CharStuff	Low-level character utilities	\	used in previous projects
StringStuff	Low-level string utilities		
MyTerminal	Low-level terminal I/O utilities		
LexAnStuff	Toolkit for building lexical analyzers	/	
Token	Token, tokenList and typeType data types		
Symbol	Symbol, symbolList and related data types		
Node	Node and related data types		
Init	Initialization of compiler		
TypeChecker	Procedures to do type-checking		
LexAn	Lexical analyzer for compiler		
SymbolTable	Compiler symbol table		
CodeWrite	Actual output of code		
CodeGen	Code generation		
ExprParser	Parses expressions		
Routines	Parses procedure and function declarations		
Parser	Main parser		

The module Debug, also supplied, is not used by the compiler, but contains routines useful in debugging the compiler.

I would appreciate hearing about any bugs in the code. My BIX name is jba. --Jonathan Amsterdam

*)

```

FROM InOut IMPORT OpenInput, OpenOutput, CloseInput, CloseOutput;
FROM MyTerminal IMPORT ClearScreen, pause, WriteLnString;
FROM Parser IMPORT program;
IMPORT Init;

```

```

BEGIN
  ClearScreen;
  WriteLnString("SIMPL Compiler V2.0");
  OpenInput('SMP');
  OpenOutput('ASM');
  Init.enterKeywords;
  program;
  CloseInput;

```



```

    CloseOutput;
    pause('Done--');
END Compiler.

```

```

=====
Start ExprParser.DEF
=====

```

```

DEFINITION MODULE ExprParser;

```

```

(* The part of the parser that handles expressions.
   Syntax:

```

```

<expr> ::= <expr> | <relexpr> | <relexpr> OR <expr> | <relexpr> AND <expr>
<relexpr> ::= <intexpr> | <intexpr> <relation> <intexpr>
<intexpr> ::= <term> | <term> + <intexpr> | <term> - <intexpr>
<term> ::= <factor> | <factor> * <term> | <factor> / <term>
<factor> ::= <idOrIndex> | <number> | <call> | <char>
              - <factor> | NOT <factor> | ( <expr> ) | <string>
<idOrIndex> ::= <id> | <idOrIndex> [ <expr> ] | <id> [ <exprlist> ]
*)

```

```

FROM Node IMPORT node;
FROM Symbol IMPORT symbol;
EXPORT QUALIFIED expr, idOrIndex;

```

```

PROCEDURE expr():node;
PROCEDURE idOrIndex(s:symbol):node;

```

```

END ExprParser.

```

```

=====
Start ExprParser.MOD
=====

```

```

IMPLEMENTATION MODULE ExprParser;

```

```

(* Handles parsing of expressions, which are tricky because we have to
   make the operators left-associative, whereas the normal recursive descent
   grammar would have them be right-associative.

```

```

   The problem is that the trees are build from the right. To make
   them get built from the left, we pass to expr, relexpr, intexpr and term
   the partial tree constructed from the left, and each of these procedures
   hooks that tree on to the one it parses in the appropriate way.

```

```

Changes for part 3:

```

1. Type coercion functions are handled.
2. Strings handled.
3. Array indexing handled.

```

*)

```

```

FROM Token IMPORT token, tokenClass, isRelation;
FROM LexAn IMPORT getToken, ungetToken, tokenErrorCheck, compError,
    getTokenClass, peekTokenClass;
FROM Symbol IMPORT symbol, Class, modeType;
IMPORT Symbol;
FROM SymbolTable IMPORT findSymbol, tInteger;
FROM Node IMPORT node, emptyNode, makeOpNode, makeUnopNode, makeIntegerNode,
    makeBooleanNode, makeSymbolNode, makeCallNode, makeStringNode, nodeEmpty,
    makeCharNode, nodeFirst, nodeRest, makeIndexNode, setNodeType, nodeType;
FROM Parser IMPORT actuals;
FROM TypeChecker IMPORT indexCheck;

```

```

CONST dummy = Period;

```

```

(* <expr> ::= <expr> | <relexpr> | <relexpr> OR <expr> | <relexpr> AND <expr> *)
PROCEDURE expr():node;

```

```

BEGIN
    RETURN expr1(emptyNode, dummy);
END expr;

```

```

PROCEDURE expr1(left:node; op:tokenClass):node;
VAR n:node;
    t:token;

```

(continued)

```

BEGIN
  n := relexpr();
  getToken(t);
  IF (t.class = And) OR (t.class = Or) THEN
    RETURN expr1(buildTree(op, left, n), t.class);
  ELSE
    ungetToken;
    IF nodeEmpty(left) THEN
      RETURN n;
    ELSE
      RETURN makeOpNode(op, left, n);
    END;
  END;
END;
END expr1;

(* <relexpr> ::= <intexpr> | <intexpr> <relation> <intexpr> *)
PROCEDURE relexpr():node;
(* Here we don't have to worry about associativity since relations aren't
   associative! *)
VAR n:node;
    t:token;
BEGIN
  n := intexpr(emptyNode, dummy);
  getToken(t);
  IF isRelation(t.class) THEN
    RETURN makeOpNode(t.class, n, intexpr(emptyNode, dummy));
  ELSE
    ungetToken;
    RETURN n;
  END;
END relexpr;

(* <intexpr> ::= <term> | <term> + <intexpr> | <term> - <intexpr> *)
PROCEDURE intexpr(left:node; op:tokenClass):node;
VAR n:node;
    t:token;
BEGIN
  n := term(emptyNode, dummy);
  getToken(t);
  IF (t.class = Plus) OR (t.class = Minus) THEN
    RETURN intexpr(buildTree(op, left, n), t.class);
  ELSE
    ungetToken;
    IF nodeEmpty(left) THEN
      RETURN n;
    ELSE
      RETURN makeOpNode(op, left, n);
    END;
  END;
END;
END intexpr;

(* <term> ::= <factor> | <factor> * <term> | <factor> / <term> *)
PROCEDURE term(left:node; op:tokenClass):node;
VAR n:node;
    t:token;
BEGIN
  n := factor();
  getToken(t);
  IF (t.class = Times) OR (t.class = Divide) THEN
    RETURN term(buildTree(op, left, n), t.class);
  ELSE
    ungetToken;
    IF nodeEmpty(left) THEN
      RETURN n;
    ELSE
      RETURN makeOpNode(op, left, n);
    END;
  END;
END;
END term;

(* <factor> ::= <id> | <number> | <call> | <char> | - <factor> | NOT <factor> |
   ( <expr> ) | <string> | <index> *)
PROCEDURE factor():node;
VAR n:node;
    t:token;
BEGIN
  getToken(t);

```



```

CASE t.class OF
  Int: RETURN makeIntegerNode(t.integer);
  Character: RETURN makeCharNode(t.ch);
  String: RETURN makeStringNode(t.string);
  Minus: RETURN makeUnopNode(UMinus, factor());
  Not: RETURN makeUnopNode(Not, factor());
  True: RETURN makeBooleanNode(TRUE);
  False: RETURN makeBooleanNode(FALSE);
  Lparen:
    n := expr();
    tokenErrorCheck(Rparen, 'Right paren expected');
    RETURN n;
  Identifier: RETURN Id(t);
ELSE
  compError('bad factor');
  RETURN emptyNode;
END;
END factor;

PROCEDURE Id(t:token):node;
VAR s:symbol;
BEGIN
  s := findSymbol(t.string);
  CASE Symbol.class(s) OF
    Func: RETURN makeCallNode(s, actuals());
    Proc: compError('procedures cannot be used in an expression');
          RETURN emptyNode;
    ArrayType, ScalarType: RETURN typeCoerce(s, actuals());
    Global, Local, Formal: (* it's a variable or index*)
      IF (Symbol.class(s) = Formal) AND (Symbol.mode(s) = mOut) THEN
        compError("can't use an OUT formal in an expression");
      END;
      RETURN idOrIndex(s);
    ELSE (* ignore *)
      RETURN emptyNode;
    END;
  END Id;

(* <idOrIndex> ::= <id> | <idOrIndex> [ <expr> ]
   That's the "official" syntax. It's easier to treat it as:
   <idOrIndex> ::= <id> <indices>
   <indices> ::= <empty> | [ <exprOrExprlist> ] <indices>
   The beginning <id> has already been read and looked up. *)
PROCEDURE idOrIndex(s:symbol):node;
BEGIN
  IF (peekTokenClass() = Lbracket) AND
    (Symbol.class(Symbol.type(s)) <> ArrayType) THEN
    compError("array variable expected");
  END;
  RETURN indices(makeSymbolNode(s));
END idOrIndex;

PROCEDURE indices(n:node):node;
BEGIN
  IF getTokenClass() = Lbracket THEN (* array index *)
    RETURN indices1(n);
  ELSE (* no index *)
    ungetToken;
    RETURN n;
  END;
END indices;

PROCEDURE indices1(n:node):node;
(* Like indices, except it starts with expression instead of Lbracket *)
VAR exp:node;
    tc:tokenClass;
BEGIN
  IF Symbol.class(nodeType(n)) <> ArrayType THEN
    compError("too many indices for array");
  END;
  exp := expr();
  indexCheck(exp);
  tc := getTokenClass();
  IF tc = Rbracket THEN
    RETURN indices(makeIndexNode(n, exp));
  ELSIF tc = Comma THEN

```

(continued)

```

        RETURN indices1(makeIndexNode(n, exp));
    ELSE
        compError("right bracket or comma expected");
        RETURN emptyNode;
    END;
END indices1;

PROCEDURE buildTree(op:tokenClass; n1, n2:node):node;
(* This is the key hack that builds trees up from the left if necessary. *)
BEGIN
    IF nodeEmpty(n1) THEN
        RETURN n2;
    ELSE
        RETURN makeOpNode(op, n1, n2);
    END;
END buildTree;

PROCEDURE typeCoerce(typeObject:symbol; actual:node):node;
(* Changes the type of the actual to typeObject. actual should be a list
of one thing. The actual's type should have the same size as the
typeObject. Note that you can do some pretty evil things with this
ability, like transforming an array[1..10][1..2] into an
array[1..2][1..10]. *)
BEGIN
    IF nodeEmpty(actual) THEN
        compError('type coercion functions must take an argument');
        RETURN actual;
    ELSE
        IF NOT nodeEmpty(nodeRest(actual)) THEN
            compError('type coercion functions take only one argument');
            (* ...but we'll set the first argument anyway *)
        END;
        IF Symbol.size(typeObject) <> Symbol.size(nodeType(nodeFirst(actual)))
        THEN compError('types not of same size');
        END;
        setNodeType(nodeFirst(actual), typeObject);
        RETURN nodeFirst(actual);
    END;
END typeCoerce;

BEGIN
END ExprParser.

=====
Start Init.DEF
=====

DEFINITION MODULE Init;

EXPORT QUALIFIED enterKeywords;

PROCEDURE enterKeywords;

END Init.

=====
Start Init.MOD
=====

IMPLEMENTATION MODULE Init;

FROM SymbolTable IMPORT enterKeyword;
FROM Token IMPORT tokenClass;

PROCEDURE enterKeywords;
BEGIN
    enterKeyword('AND', And);
    enterKeyword('ARRAY', Array);
    enterKeyword('BEGIN', Begin);
    enterKeyword('DO', Do);
    enterKeyword('ELSE', Else);
    enterKeyword('ELSIF', Elsif);
    enterKeyword('END', End);

```



```

enterKeyword('FALSE', False);
enterKeyword('FUNCTION', Function);
enterKeyword('IF', If);
enterKeyword('IN', In);
enterKeyword('NOT', Not);
enterKeyword('OF', Of);
enterKeyword('OR', Or);
enterKeyword('OUT', Out);
enterKeyword('PROCEDURE', Procedure);
enterKeyword('PROGRAM', Program);
enterKeyword('READ', Read);
enterKeyword('RETURN', Return);
enterKeyword('THEN', Then);
enterKeyword('TRUE', True);
enterKeyword('TYPE', Type);
enterKeyword('VAR', Var);
enterKeyword('WHILE', While);
enterKeyword('WRITE', Write);
END enterKeywords;

BEGIN
END Init.

=====
Start LexAn.DEF
=====

DEFINITION MODULE LexAn;

(* The lexical analyzer for the SIMPL compiler. It uses the InOut module
   do input, so you can get input from a file by redirecting it with
   InOut.OpenInput.
   This module also handles errors. *)

FROM Token IMPORT token, tokenClass;

EXPORT QUALIFIED getToken, ungetToken, getTokenClass, tokenErrorCheck,
   getTokenErrorCheck, errorFlag, compError, peekTokenClass;

VAR errorFlag:BOOLEAN;      (* Set to TRUE when an error occurs. *)

PROCEDURE getToken(VAR t:token);
(* Get a token from the input stream. *)

PROCEDURE ungetToken;
(* Push a token back on the input stream. Can only unget one at a time. *)

PROCEDURE getTokenClass():tokenClass;
(* Get a token from the input stream, but just return its class. *)

PROCEDURE peekTokenClass():tokenClass;
(* Get a token from the input stream, unget it, and return its class. *)

PROCEDURE tokenErrorCheck(tc:tokenClass; msg: ARRAY OF CHAR);
(* Read a token from the input stream and compare its class to tc. If they
   are the same, do nothing. If they are different, write the current line
   to the screen, print the message and unget the token. *)

PROCEDURE getTokenErrorCheck(VAR t:token; tc:tokenClass; msg: ARRAY OF CHAR);
(* Like tokenErrorCheck, but returns the token as well. *)

PROCEDURE compError(msg:ARRAY OF CHAR);
(* Writes the current line and displays msg. Sets errorFlag to TRUE. *)

END LexAn.

=====
Start LexAn.MOD
=====

IMPLEMENTATION MODULE LexAn;

(* Lexical analyzer for the SIMPL compiler. Uses the routines in LexAnStuff.*)

```

(continued)

```

FROM InOut IMPORT EOL;
FROM Token IMPORT token, tokenClass;
FROM MyTerminal IMPORT fatal, WriteLnString;
FROM StringStuff IMPORT stringLen;
FROM SymbolTable IMPORT findKeyword;
FROM LexAnStuff IMPORT dispatch, enterAll, enterChar, enterEndOfFile, ignore,
    enterAlphas, enterDigits, skipToChar, alphaNumString, posInteger, string,
    enterWhite, writeLine, getChar, ungetChar;

```

```

VAR tok: token;
    ungotten: BOOLEAN;

```

```

PROCEDURE getToken(VAR t:token);
BEGIN
    getTok;
    t := tok;
END getToken;

```

```

PROCEDURE getTok;
VAR c:CHAR;
BEGIN
    IF ungotten THEN
        ungotten := FALSE;
    ELSE
        dispatch;
    END;
END getTok;

```

```

PROCEDURE ungetToken;
BEGIN
    IF ungotten THEN
        fatal("ungetToken: can only unget one token at a time");
    ELSE
        ungotten := TRUE;
    END;
END ungetToken;

```

```

PROCEDURE getTokenClass():tokenClass;
BEGIN
    getTok;
    RETURN tok.class;
END getTokenClass;

```

```

PROCEDURE peekTokenClass():tokenClass;
BEGIN
    getTok;
    ungetToken;
    RETURN tok.class;
END peekTokenClass;

```

```

PROCEDURE tokenErrorCheck(tc:tokenClass; msg: ARRAY OF CHAR);
BEGIN
    getTok;
    IF tok.class <> tc THEN
        compError(msg);
        IF tok.class = EndOfFile THEN
            fatal("unexpected end of input");
        END;
        ungetToken;
    END;
END tokenErrorCheck;

```

```

PROCEDURE getTokenErrorCheck(VAR t:token; tc:tokenClass; msg: ARRAY OF CHAR);
BEGIN
    tokenErrorCheck(tc, msg);
    t := tok;
END getTokenErrorCheck;

```

(*** reading procedures ***)

```

PROCEDURE illegalChar(c:CHAR);
VAR charstring:ARRAY[0..1] OF CHAR;
BEGIN
    charstring[0] := c;          (* fake a 1-char string *)
    charstring[1] := 0C;
    compError('illegal character');
    getTok;

```



```

END illegalChar;

PROCEDURE comment(c:CHAR);      (* Comments are ignored. They are delimited
                                by { and } *)
BEGIN
    skipToChar('}');
    getTok;
END comment;

PROCEDURE idOrKeyword(c:CHAR);
(* Get an alphanumeric string from the input. If we find it in the symbol
   table marked as a keyword, then it's a keyword; findKeyword will have taken
   care of setting tok.class to the right value. Else, it's an identifier. *)
BEGIN
    IF NOT alphaNumString(c, tok.string) THEN
        compError('identifier too long');
    END;
    IF NOT findKeyword(tok.string, tok.class) THEN
        tok.class := Identifier;
    END;
END idOrKeyword;

PROCEDURE posInt(c:CHAR);
BEGIN
    tok.class := Int;
    tok.integer := posInteger(c);
END posInt;

PROCEDURE charProc(c:CHAR);
(* Read a character, delimited by delim, from the input. Can use
   backslash: \n = newline, \t = tab, anything else literal. *)
BEGIN
    tok.class := Character;
    IF (NOT string(c, tok.string)) OR (stringLen(tok.string) > 1) THEN
        compError('illegal character constant');
    END;
    tok.ch := tok.string[0];
END charProc;

PROCEDURE stringProc(delim:CHAR);
(* Read a string from the input. If too long, skip to the next delim. *)
VAR i:CARDINAL;
    c:CHAR;
BEGIN
    tok.class := String;
    IF NOT string(delim, tok.string) THEN
        compError('string too long');
        skipToChar(delim);
        delim := getChar();      (* get the delimiter *)
    END;
END stringProc;

(** Reading special characters **)

PROCEDURE semicolon(c:CHAR); BEGIN tok.class := Semicolon; END semicolon;
PROCEDURE equal(c:CHAR); BEGIN tok.class := Equal; END equal;
PROCEDURE comma(c:CHAR); BEGIN tok.class := Comma; END comma;
PROCEDURE plus(c:CHAR); BEGIN tok.class := Plus; END plus;
PROCEDURE minus(c:CHAR); BEGIN tok.class := Minus; END minus;
PROCEDURE times(c:CHAR); BEGIN tok.class := Times; END times;
PROCEDURE divide(c:CHAR); BEGIN tok.class := Divide; END divide;
PROCEDURE lparen(c:CHAR); BEGIN tok.class := Lparen; END lparen;
PROCEDURE rparen(c:CHAR); BEGIN tok.class := Rparen; END rparen;
PROCEDURE lbracket(c:CHAR); BEGIN tok.class := Lbracket; END lbracket;
PROCEDURE rbracket(c:CHAR); BEGIN tok.class := Rbracket; END rbracket;

PROCEDURE greater(c:CHAR);
BEGIN
    IF getChar() = '=' THEN
        tok.class := GreaterEqual;
    ELSE
        ungetChar;
        tok.class := Greater;
    END;

```

(continued)

```

END;
END greater;

PROCEDURE less(c:CHAR);
BEGIN
  c := getChar();
  IF c = '=' THEN
    tok.class := LessEqual;
  ELSIF c = '>' THEN
    tok.class := NotEqual;
  ELSE
    ungetChar;
    tok.class := Less;
  END;
END less;

PROCEDURE colon(c:CHAR);
BEGIN
  IF getChar() = '=' THEN
    tok.class := Assignment;
  ELSE
    ungetChar;
    tok.class := Colon;
  END;
END colon;

PROCEDURE periodOrDotdot(c:CHAR);
BEGIN
  IF getChar() = '.' THEN
    tok.class := DotDot;
  ELSE
    ungetChar;
    tok.class := Period;
  END;
END periodOrDotdot;

PROCEDURE endOfInput(c:CHAR);
BEGIN
  tok.class := EndOfInput;
END endOfInput;

(** initialization of charTable **)

PROCEDURE initCharTable;
BEGIN
  enterAll(illegalChar);
  enterWhite(ignore);
  enterAlphas(idOrKeyword);
  enterDigits(posInt);
  enterChar('.', periodOrDotdot);
  enterChar(':', colon);
  enterChar(';', semicolon);
  enterChar('(', lparen);
  enterChar(')', rparen);
  enterChar(',', comma);
  enterChar('=', equal);
  enterChar('>', greater);
  enterChar('<', less);
  enterChar('+', plus);
  enterChar('-', minus);
  enterChar('*', times);
  enterChar('/', divide);
  enterChar('{', comment);
  enterChar('"', stringProc);
  enterChar("'", charProc);
  enterChar('[', lbracket);
  enterChar(']', rbracket);
  enterEndOfFile(endOfInput);
END initCharTable;

(** errors **)

PROCEDURE compError(msg:ARRAY OF CHAR);
BEGIN
  writeLine;
  WriteLnString(msg);
  errorFlag := TRUE;

```



```
END compError;
```

```
BEGIN
  ungotten := FALSE;
  errorFlag := FALSE;
  initCharTable;
END LexAn.
```

```
=====
Start Node.DEF
=====
```

```
DEFINITION MODULE Node;
```

```
(* Nodes are what make up the parse tree produced by the SIMPL parser.
   They consist of all data relevant to generating code.
```

```
Changes for part 3:
```

```
1. setNodeType added so ExprParser.typeCoerce could change it.
```

```
*)
```

```
FROM Token IMPORT tokenClass;
FROM Symbol IMPORT symbol;
```

```
EXPORT QUALIFIED node, nodeClass, NodeClass, emptyNode, nodeEmpty, freeNode,
  makeStmtsNode, makeIfNode, makeWhileNode, makeReturnNode,
  makeAssignmentNode, makeExprListNode, makeOpNode, makeUnopNode,
  makeIntegerNode, makeBooleanNode, makeSymbolNode, makeCallNode,
  makeWriteNode, makeReadNode, makeStringNode, makeCharNode,
  nodeFirst, nodeRest, nodeTest, nodeThen, nodeElse, nodeStmts, nodeRHS,
  nodeLHS, nodeArgs, nodeRoutine, nodeExpr, nodeArg, nodeLeftArg,
  nodeRightArg, nodeOp, nodeSymbol, nodeType, nodeInt, nodeBool,
  nodeNumFormals, nodeString, nodeChar, setNodeType, nodeArray, nodeIndex,
  makeIndexNode;
```

```
TYPE
  NodeClass = (* the different kinds of nodes *)
    (nOp, (* binary operators (+, -, *, /, relations, AND, OR) *)
     nUnop, (* unary operators (unary minus, NOT) *)
     nBool, (* a boolean constant (TRUE, FALSE) *)
     nInt, (* an integer constant *)
     nChar, (* a character constant *)
     nString, (* a string constant *)
     nSymbol, (* a symbol (variable) *)
     nIf, (* IF statement *)
     nWhile, (* WHILE statement *)
     nReturn, (* RETURN statement, either procedure or function *)
     nCall, (* procedure call (statement) or function call *)
     nAssignment, (* assignment statement *)
     nWrite, nRead, (* WRITE and READ statements *)
     nList, (* a list of statements or expressions *)
     nIndex); (* an array index *)
```

```
node;
```

```
VAR emptyNode: node;
```

```
PROCEDURE nodeClass(n:node):NodeClass;
(* Returns the class of node *)
```

```
PROCEDURE nodeEmpty(n:node):BOOLEAN;
(* Returns true if node is the emptyNode *)
```

```
PROCEDURE freeNode(n:node);
(* Frees the storage associated with n *)
```

```
(*** Node creation ***)
```

```
PROCEDURE makeStmtsNode(first, rest:node):node;
(* Make a node representing a list of statements *)
```

```
PROCEDURE makeReturnNode(routine:symbol; returnExpr:node):node;
(* Make a return node. Routine is the routine we are returning from.
   returnExpr is an expression to be returned, for functions; for procedures,
```

(continued)

it should be the empty node. *)

```
PROCEDURE makeCallNode(name:symbol; actuals:node):node;
PROCEDURE makeWriteNode(actuals:node):node;
PROCEDURE makeReadNode(actuals:node):node;
(* In all of these, actuals should have been made with makeExprListNode. *)
```

```
PROCEDURE makeIfNode(test, then, else:node):node;
PROCEDURE makeWhileNode(test, stmts:node):node;
PROCEDURE makeAssignmentNode(lhs, expr:node):node;
PROCEDURE makeExprListNode(first, rest:node):node;
PROCEDURE makeOpNode(op:tokenClass; leftarg, rightarg:node):node;
PROCEDURE makeUnopNode(op:tokenClass; arg:node):node;
PROCEDURE makeIntegerNode(i:INTEGER):node;
PROCEDURE makeBooleanNode(b:BOOLEAN):node;
PROCEDURE makeSymbolNode(id:symbol):node;
PROCEDURE makeStringNode(s:ARRAY OF CHAR):node;
PROCEDURE makeCharNode(c:CHAR):node;
PROCEDURE makeIndexNode(array, index:node):node;
```

(*** Accessing parts of nodes **)

(* many nodes have a type associated with them *)

```
PROCEDURE nodeType(n:node):symbol;
PROCEDURE setNodeType(n:node; typeObject:symbol);
```

(* for constants *)

```
PROCEDURE nodeInt(n:node):INTEGER;
PROCEDURE nodeBool(n:node):BOOLEAN;
PROCEDURE nodeChar(n:node):CHAR;
PROCEDURE nodeString(n:node; VAR s:ARRAY OF CHAR);
(* Just truncates if s is too short. *)
```

(* for lists *)

```
PROCEDURE nodeFirst(n:node):node;
PROCEDURE nodeRest(n:node):node;
```

(* for IF statements *)

```
PROCEDURE nodeTest(n:node):node; (* also for WHILE statements *)
PROCEDURE nodeThen(n:node):node;
PROCEDURE nodeElse(n:node):node;
```

(* for WHILE statements *)

```
PROCEDURE nodeStmts(n:node):node;
```

(* for assignment statements *)

```
PROCEDURE nodeRHS(n:node):node; (* right-hand side *)
PROCEDURE nodeLHS(n:node):node; (* left-hand side *)
```

(* for calls *)

```
PROCEDURE nodeArgs(n:node):node;
PROCEDURE nodeRoutine(n:node):symbol;
```

(* for RETURN statements *)

```
PROCEDURE nodeExpr(n:node):node;
PROCEDURE nodeNumFormals(n:node):CARDINAL;
```

(* for ops and unops *)

```
PROCEDURE nodeArg(n:node):node;
PROCEDURE nodeLeftArg(n:node):node;
PROCEDURE nodeRightArg(n:node):node;
PROCEDURE nodeOp(n:node):tokenClass;
```

(* for symbols *)

```
PROCEDURE nodeSymbol(n:node):symbol;
```

(* for array indexing *)

```
PROCEDURE nodeArray(n:node):node;
PROCEDURE nodeIndex(n:node):node;
```

END Node.

```
=====
Start Node.MOD
=====
```

IMPLEMENTATION MODULE Node;


```
(* Procedures for constructing and manipulating the nodes of the parse tree.
Most type-checking is done here. *)
```

```
FROM Token IMPORT tokenClass, stringType, isRelation;
FROM Symbol IMPORT symbol, Class, emptySymbol, numFormals, symbolList,
    s1Symbol, s1Empty, s1Next;
IMPORT Symbol;
FROM SymbolTable IMPORT tUnknown, tString, tInteger, tChar, tBoolean;
FROM Storage IMPORT ALLOCATE, DEALLOCATE;
FROM TypeChecker IMPORT typeCompatible, opAppropriate, callCheck, unopCheck,
    readCheck, writeCheck, binopCheck, returnCheck, assignCheck, baseType;
FROM MyTerminal IMPORT WriteString, fatal;
FROM StringStuff IMPORT stringCopy;
```

```
TYPE
```

```
node = POINTER TO nodeRec;
nodeRec = RECORD
    type: symbol;
    CASE class:NodeClass OF
        nOp: op: tokenClass; leftArg, rightArg: node;
        nUnop: unop: tokenClass; arg: node;
        nBool: bool: BOOLEAN;
        nInt: int: INTEGER;
        nChar: ch: CHAR;
        nString: string:stringType;
        nSymbol: sym: symbol;
        nIf: test, then, else: node;
        nWhile: wtest, stmts: node;
        nAssignment: LHS, RHS:node;
        nCall, nWrite, nRead: routine:symbol; args:node;
        nReturn: nFormals:CARDINAL; expr:node;
        nList: first, rest:node;
        nIndex: array, index:node;
    END;
END;
```

```
PROCEDURE nodeClass(n:node):NodeClass;
BEGIN
    RETURN n^.class;
END nodeClass;
```

```
PROCEDURE nodeEmpty(n:node):BOOLEAN;
BEGIN
    RETURN n = emptyNode;
END nodeEmpty;
```

```
PROCEDURE freeNode(n:node);
BEGIN
    IF n <> emptyNode THEN
        WITH n^ DO CASE class OF
            nInt, nBool, nSymbol, nString, nChar: (* do nothing *);
            | nOp: freeNode(leftArg);
            | freeNode(rightArg);
            | nUnop: freeNode(arg);
            | nIf: freeNode(test);
            | freeNode(then);
            | freeNode(else);
            | nWhile: freeNode(wtest);
            | freeNode(stmts);
            | nAssignment: freeNode(LHS);
            | freeNode(RHS);
            | nCall, nRead, nWrite: freeNode(args);
            | nReturn: freeNode(expr);
            | nList: freeNode(first);
            | freeNode(rest);
            | nIndex: freeNode(array);
            | freeNode(index);
        ELSE
            WriteString("freeNode: unknown node type");
        END; END;
        DISPOSE(n); (* , n^.class); *)
    END;
END freeNode;
```

(continued)

(*** node creation ***)

```
PROCEDURE makeStmtsNode(first, rest:node):node;
VAR n:node;
BEGIN
  n := newNode(nList);
  n^.first := first;
  n^.rest := rest;
  RETURN n;
END makeStmtsNode;
```

```
PROCEDURE makeExprListNode(first, rest:node):node;
VAR n:node;
BEGIN
  n := newNode(nList);
  n^.first := first;
  n^.rest := rest;
  RETURN n;
END makeExprListNode;
```

```
PROCEDURE makeIfNode(test, then, else:node):node;
VAR n:node;
BEGIN
  n := newNode(nIf);
  n^.test := test;
  n^.then := then;
  n^.else := else;
  RETURN n;
END makeIfNode;
```

```
PROCEDURE makeWhileNode(test, stmts:node):node;
VAR n:node;
BEGIN
  n := newNode(nWhile);
  n^.wtest := test;
  n^.stmts := stmts;
  RETURN n;
END makeWhileNode;
```

```
PROCEDURE makeReturnNode(routine:symbol; returnExpr:node):node;
VAR n:node;
BEGIN
  n := newNode(nReturn);
  n^.expr := returnExpr;
  IF returnCheck(routine, returnExpr) THEN
    n^.nFormals := sizeFormals(routine);
  END;
  RETURN n;
END makeReturnNode;
```

```
PROCEDURE sizeFormals(routine:symbol):CARDINAL;
(* Returns the number of words occupied by formals. Before this was just
the number of formals, but now each open array param adds 2 to the count.
(For its bounds.) *)
VAR formlist:symbolList;
sum:CARDINAL;
t:symbol;
BEGIN
  formlist := Symbol.formals(routine);
  sum := 0;
  WHILE NOT sIsEmpty(formlist) DO
    t := Symbol.type(s!Symbol(formlist));
    IF (Symbol.class(t) = ArrayType) AND Symbol.open(t) THEN
      INC(sum, 3);
    ELSE
      INC(sum);
    END;
    formlist := s!Next(formlist);
  END;
  RETURN sum;
END sizeFormals;
```

```
PROCEDURE makeAssignmentNode(lhs, expr:node):node;
VAR n:node;
BEGIN
  n := newNode(nAssignment);
```



```

    n^.LHS := lhs;
    n^.RHS := expr;
    assignCheck(lhs, expr);
    RETURN n;
END makeAssignmentNode;

PROCEDURE makeOpNode(op:tokenClass; leftarg, rightarg:node):node;
VAR n:node;
    typeOK:BOOLEAN;
BEGIN
    n := newNode(nOp);
    n^.op := op;
    n^.leftArg := leftarg;
    n^.rightArg := rightarg;
    typeOK := binopCheck(op, leftarg, rightarg);
    IF isRelation(op) THEN
        n^.type := tBoolean;
    ELSIF typeOK THEN
        n^.type := leftarg^.type;
    ELSE
        n^.type := tUnknown;
    END;
    RETURN n;
END makeOpNode;

PROCEDURE makeUnopNode(op:tokenClass; arg:node):node;
VAR n:node;
BEGIN
    n := newNode(nUnop);
    n^.unop := op;
    n^.arg := arg;
    IF unopCheck(op, arg) THEN
        n^.type := arg^.type;
    ELSE
        n^.type := tUnknown;
    END;
    RETURN n;
END makeUnopNode;

PROCEDURE makeIntegerNode(i:INTEGER):node;
VAR n:node;
BEGIN
    n := newNode(nInt);
    n^.type := tInteger;
    n^.int := i;
    RETURN n;
END makeIntegerNode;

PROCEDURE makeBooleanNode(b:BOOLEAN):node;
VAR n:node;
BEGIN
    n := newNode(nBool);
    n^.type := tBoolean;
    n^.bool := b;
    RETURN n;
END makeBooleanNode;

PROCEDURE makeCharNode(c:CHAR):node;
VAR n:node;
BEGIN
    n := newNode(nChar);
    n^.type := tChar;
    n^.ch := c;
    RETURN n;
END makeCharNode;

PROCEDURE makeSymbolNode(s:symbol):node;
VAR n:node;
BEGIN
    n := newNode(nSymbol);
    n^.type := Symbol.type(s);
    n^.sym := s;
    RETURN n;
END makeSymbolNode;

```

(continued)

```

PROCEDURE makeCallNode(name:symbol; actuals:node):node;
VAR n:node;
BEGIN
  n := newNode(nCall);
  WITH n^ DO
    routine := name;
    args := actuals;
    type := Symbol.type(name);
    callCheck(routine, args);
  END;
  RETURN n;
END makeCallNode;

```

```

PROCEDURE makeWriteNode(actuals:node):node;
VAR n:node;
BEGIN
  writeCheck(actuals);
  n := newNode(nWrite);
  n^.routine := emptySymbol;
  n^.args := actuals;
  RETURN n;
END makeWriteNode;

```

```

PROCEDURE makeReadNode(actuals:node):node;
VAR n:node;
BEGIN
  readCheck(actuals);
  n := newNode(nRead);
  n^.routine := emptySymbol;
  n^.args := actuals;
  RETURN n;
END makeReadNode;

```

```

PROCEDURE makeStringNode(s:ARRAY OF CHAR):node;
VAR n:node;
BEGIN
  n := newNode(nString);
  stringCopy(n^.string, s);
  n^.type := tString;
  RETURN n;
END makeStringNode;

```

```

PROCEDURE makeIndexNode(array, index:node):node;
VAR n:node;
BEGIN
  n := newNode(nIndex);
  IF Symbol.empty(nodeType(array)) THEN
    n^.type := emptySymbol;
  ELSE
    n^.type := baseType(Symbol.type(nodeType(array)));
  END;
  n^.array := array;
  n^.index := index;
  RETURN n;
END makeIndexNode;

```

```

PROCEDURE newNode(nc:NodeClass):node;
VAR n:node;
BEGIN
  NEW(n); (* nc; *)
  n^.class := nc;
  n^.type := tUnknown;
  RETURN n;
END newNode;

```

(*** node access ***)

```

PROCEDURE nodeInt(n:node):INTEGER;
BEGIN
  nodeClassCheck('nodeInt', n, nInt);
  RETURN n^.int;
END nodeInt;

```

```

PROCEDURE nodeBool(n:node):BOOLEAN;
BEGIN
  nodeClassCheck('nodeBool', n, nBool);

```



```

    RETURN n^.bool;
END nodeBool;

PROCEDURE nodeChar(n:node):CHAR;
BEGIN
    nodeClassCheck('nodeChar', n, nChar);
    RETURN n^.ch;
END nodeChar;

PROCEDURE nodeString(n:node; VAR s:ARRAY OF CHAR);
BEGIN
    nodeClassCheck('nodeString', n, nString);
    stringCopy(s, n^.string);
END nodeString;

PROCEDURE nodeFirst(n:node):node;
BEGIN
    nodeClassCheck('nodeFirst', n, nList);
    RETURN n^.first;
END nodeFirst;

PROCEDURE nodeRest(n:node):node;
BEGIN
    nodeClassCheck('nodeRest', n, nList);
    RETURN n^.rest;
END nodeRest;

PROCEDURE nodeTest(n:node):node;
BEGIN
    IF n^.class = nIf THEN
        RETURN n^.test;
    ELSIF n^.class = nWhile THEN
        RETURN n^.wtest;
    ELSE
        nodeClassError('nodeTest');
        RETURN emptyNode;
    END;
END nodeTest;

PROCEDURE nodeThen(n:node):node;
BEGIN
    nodeClassCheck('nodeThen', n, nIf);
    RETURN n^.then;
END nodeThen;

PROCEDURE nodeElse(n:node):node;
BEGIN
    nodeClassCheck('nodeElse', n, nIf);
    RETURN n^.else;
END nodeElse;

PROCEDURE nodeStmts(n:node):node;
BEGIN
    nodeClassCheck('nodeStmts', n, nWhile);
    RETURN n^.stmts;
END nodeStmts;

PROCEDURE nodeRHS(n:node):node;
BEGIN
    nodeClassCheck('nodeRHS', n, nAssignment);
    RETURN n^.RHS;
END nodeRHS;

PROCEDURE nodeLHS(n:node):node;
BEGIN
    nodeClassCheck('nodeLHS', n, nAssignment);
    RETURN n^.LHS;
END nodeLHS;

PROCEDURE nodeArgs(n:node):node;
BEGIN
    WITH n^ DO
        IF (class = nCall) OR (class = nRead) OR (class = nWrite) THEN
            RETURN args;
        ELSE

```

(continued)

```

        nodeClassError('nodeArgs');
    END;
END nodeArgs;

PROCEDURE nodeRoutine(n:node):symbol;
BEGIN
    nodeClassCheck('nodeRoutine', n, nCall);
    RETURN n^.routine;
END nodeRoutine;

PROCEDURE nodeExpr(n:node):node;
BEGIN
    nodeClassCheck('nodeExpr', n, nReturn);
    RETURN n^.expr;
END nodeExpr;

PROCEDURE nodeArg(n:node):node;
BEGIN
    nodeClassCheck('nodeArg', n, nUnop);
    RETURN n^.arg;
END nodeArg;

PROCEDURE nodeLeftArg(n:node):node;
BEGIN
    nodeClassCheck('nodeLeftArg', n, nOp);
    RETURN n^.leftArg;
END nodeLeftArg;

PROCEDURE nodeRightArg(n:node):node;
BEGIN
    nodeClassCheck('nodeRightArg', n, nOp);
    RETURN n^.rightArg;
END nodeRightArg;

PROCEDURE nodeOp(n:node):tokenClass;
BEGIN
    IF n^.class = nOp THEN
        RETURN n^.op;
    ELSIF n^.class = nUnop THEN
        RETURN n^.unop;
    ELSE
        nodeClassError('nodeOp');
        RETURN Plus;
    END;
END nodeOp;

PROCEDURE nodeSymbol(n:node):symbol;
BEGIN
    nodeClassCheck('nodeSymbol', n, nSymbol);
    RETURN n^.sym;
END nodeSymbol;

PROCEDURE nodeArray(n:node):node;
BEGIN
    nodeClassCheck("nodeArray", n, nIndex);
    RETURN n^.array;
END nodeArray;

PROCEDURE nodeIndex(n:node):node;
BEGIN
    nodeClassCheck("nodeIndex", n, nIndex);
    RETURN n^.index;
END nodeIndex;

PROCEDURE nodeNumFormals(n:node):CARDINAL;
BEGIN
    nodeClassCheck('nodeNumFormals', n, nReturn);
    RETURN n^.nFormals;
END nodeNumFormals;

PROCEDURE nodeType(n:node):symbol;
BEGIN
    RETURN n^.type;
END nodeType;

```



```

PROCEDURE setNodeType(n:node; typeObject:symbol);
BEGIN
    n^.type := typeObject;
END setNodeType;

(** other **)

PROCEDURE nodeClassCheck(s:ARRAY OF CHAR; n:node; nc:NodeClass);
BEGIN
    IF n^.class <> nc THEN
        nodeClassError(s);
    END;
END nodeClassCheck;

PROCEDURE nodeClassError(s:ARRAY OF CHAR);
BEGIN
    WriteString(s);
    fatal(": node of wrong type");
END nodeClassError;

BEGIN
    emptyNode := NIL;
END Node.

=====
Start Parser.DEF
=====

DEFINITION MODULE Parser;

(* This is the bulk of the SIMPL parser. It covers most of the language.
   For routines (procedures and functions) see Routines.
   For expressions, see ExprParser.

   Syntax handled by this module:

<program> ::= PROGRAM <id>; <vars> <routines> <block> .

<types> ::= <empty> | TYPE <typelist>
<typelist> ::= <typeddecl> | <typeddecl> <typelist>
<typeddecl> ::= <id> = <type> ;
<type> ::= <id> | ARRAY [ <int> .. <int> ] OF <type>

<vars> ::= <empty> | VAR <varlist>
<varlist> ::= <decl> | <decl> <varlist>
<decl> ::= <idlist> : <type> ;
<idlist> ::= <id> | <id> , <idlist>

<block> ::= BEGIN <stmts> END
<stmts> ::= <empty> | <stmt> ; <stmts>
<stmt> ::= <while> | <if> | <return> | <assign> | <call>
<while> ::= WHILE <expr> DO <stmts> END
<if> ::= IF <elsif> END
<elsif> ::= <expr> THEN <stmts> <else>
<else> ::= <empty> | ELSIF <elsif> | ELSE <stmts>
<return> ::= RETURN | RETURN <expr>
<assign> ::= <idOrindex> := <expr>
<call> ::= <id> <actuals>
<actuals> ::= <empty> | ( <exprlist> )
<exprlist> ::= <expr> | <expr> , <exprlist>
*)

FROM Symbol IMPORT symbol;
FROM Node IMPORT node;
FROM Token IMPORT tokenList;

EXPORT QUALIFIED program, types, vars, idlist, block, actuals;

PROCEDURE program;
(* Parse the entire program *)

```

(continued)

```

PROCEDURE types(routineName:symbol);
(* Parse type declarations. See below for explanation of routineName. *)

PROCEDURE vars(routineName:symbol);
(* Parse variable declarations. RoutineName is the name of the routine
currently being compiled; if these are global variables, it should be
the name of the program. *)

PROCEDURE idlist():tokenList;
(* Parse a list of identifiers *)

PROCEDURE block(routine:symbol):node;
(* Parse a block of code. Routine is the routine currently being compiled. *)

PROCEDURE actuals():node;
(* Parse a list of actual parameters, i.e. a list of expressions. *)

END Parser.

```

```

=====
Start Parser.MOD
=====

```

```
IMPLEMENTATION MODULE Parser;
```

```
(* Most of the parser for the SIMPL compiler. It is a top-down, recursive
descent parser.
```

```
Changes for part 3:
```

1. Type declarations and arrays are parsed.
2. Type objects used instead of the old typeType.

```

*)

FROM Token IMPORT token, tokenClass, emptyTokenList, stringType,
  tiToken, tiNext, tiEmpty, addToTokenList, tokenList, freeTokenList;
FROM LexAn IMPORT getToken, getTokenClass, peekTokenClass, compError,
  ungetToken, tokenErrorCheck, getTokenErrorCheck;
FROM Symbol IMPORT symbol, emptySymbol, isType, Class;
IMPORT Symbol;
FROM SymbolTable IMPORT enterSymbol, enterLocal, enterFormal, findSymbol,
  currentLexLevel, tUnknown, enterArrayType;
FROM Node IMPORT node, emptyNode, makeStmtsNode, makeIfNode, makeWhileNode,
  makeReturnNode, makeAssignmentNode, makeExprListNode,
  makeCallNode, makeReadNode, makeWriteNode, nodeType, makeSymbolNode;
FROM CodeGen IMPORT genBlock, genGlobal;
FROM CodeWrite IMPORT writeStringBranch, writeHalt, writeRoutineLabel,
  writeStrings;
FROM TypeChecker IMPORT boolCheck, assignable;
FROM ExprParser IMPORT expr, idOrIndex;
FROM Routines IMPORT routines;
FROM MyTerminal IMPORT fatal;
VAR programName:symbol;

(* <program> ::= PROGRAM <id>; <types> <vars> <routines> <block> . *)
PROCEDURE program;
VAR t:token;
  n:node;
BEGIN
  tokenErrorCheck(Program, 'keyword "PROGRAM" expected');
  getTokenErrorCheck(t, Identifier, 'name of program expected');
  IF t.class <> Identifier THEN
    t.string := "???"; (* if the program name isn't given, make one up *)
  END;
  programName := enterSymbol(t.string, Proc, tUnknown);
  writeStringBranch(t.string);
  tokenErrorCheck(Semicolon, 'semicolon expected');
  types(emptySymbol);
  vars(emptySymbol);
  routines;
  writeRoutineLabel(programName);
  genBlock(block(programName));

```



```

tokenErrorCheck(Period, 'period expected');
tokenErrorCheck(EndOfInput, 'end of input expected');
writeHalt;
writeStrings;
END program;

(** type declarations **)

(* <types> ::= <empty> | TYPE <typelist> *)
PROCEDURE types(routineName:symbol);
BEGIN
  IF getTokenClass() = Type THEN
    typelist(routineName);
  ELSE
    ungetToken;
  END;
END types;

(* <typelist> ::= <typedec1> | <typedec1> <typelist> *)
PROCEDURE typelist(routineName:symbol);
BEGIN
  typedec1(routineName);
  IF peekTokenClass() = Identifier THEN
    typelist(routineName);
  END;
END typelist;

(* <typedec1> ::= <id> = <type> ; *)
PROCEDURE typedec1(routineName:symbol);
VAR typeToken:token;
    typeObject:symbol;
BEGIN
  getToken(typeToken);
  IF typeToken.class <> Identifier THEN
    compError('identifier expected');
    ungetToken;
    typeToken.string := '???'; (* make up a type name *)
  END;
  tokenErrorCheck(Equal, 'equal sign expected');
  typeObject := type();
  IF NOT Symbol.empty(typeObject) THEN
    IF Symbol.class(typeObject) = ArrayType THEN
      handleArrayType(typeToken.string, typeObject);
    ELSIF Symbol.class(typeObject) = ScalarType THEN
      typeObject := enterSymbol(typeToken.string, ScalarType, typeObject);
    ELSE
      (* do nothing--error caught in type() *)
    END;
  END;
  tokenErrorCheck(Semicolon, "semicolon expected");
END typedec1;

PROCEDURE handleArrayType(VAR s:stringType; typeObject:symbol);
(* If the array type hasn't been named, name it and insert it; else,
   copy it and give the copy the new name. *)
VAR typeName:stringType;
BEGIN
  IF Symbol.anonymous(typeObject) THEN
    enterArrayType(s, typeObject);
  ELSE
    enterArrayType(s, Symbol.copyArrayType(typeObject));
  END;
END handleArrayType;

(* <type> ::= <id> | ARRAY [ Int .. Int ] OF <type> *)
PROCEDURE type():symbol;
VAR t:token;
    typeSymbol:symbol;
BEGIN
  getToken(t);
  IF t.class = Identifier THEN
    typeSymbol := findSymbol(t.string);
    IF isType(typeSymbol) THEN
      RETURN typeSymbol;
    ELSE

```

(continued)

```

        compError('not a type');
        RETURN tUnknown;
    END;
    ELSIF t.class = Array THEN
        tokenErrorCheck(Lbracket, "left bracket expected");
        RETURN type1();
    ELSE
        compError("Identifier or ARRAY expected");
        ungetToken;
        RETURN tUnknown;
    END;
END type;

PROCEDURE type1():symbol;
VAR t:token;
    lowBound, highBound, temp:INTEGER;
BEGIN
    lowBound := getBound();
    tokenErrorCheck(DotDot, 'two dots expected');
    highBound := getBound();
    IF lowBound > highBound THEN
        compError("lowBound > highBound");
        temp := lowBound;
        lowBound := highBound;
        highBound := temp;
    END;
    getToken(t);
    IF t.class = Comma THEN
        RETURN Symbol.newArrayType(type1(), lowBound, highBound, FALSE,
                                   currentLexLevel());
    ELSIF t.class = Rbracket THEN
        tokenErrorCheck(Of, 'OF expected');
        RETURN Symbol.newArrayType(type1(), lowBound, highBound, FALSE,
                                   currentLexLevel());
    ELSE
        compError("Comma or right bracket expected");
        ungetToken;
        RETURN tUnknown;
    END;
END type1;

PROCEDURE getBound():INTEGER;
(* Reads in an integer, possibly negative. *)
VAR t:token;
BEGIN
    getToken(t);
    IF t.class = Int THEN
        RETURN t.integer;
    ELSIF t.class = Minus THEN
        getTokenErrorCheck(t, Int, "integer expected");
        RETURN -t.integer;
    ELSE
        ungetToken;
        compError("integer expected");
        RETURN 0;
    END;
END getBound;

    (** variable declarations **)

(* <vars> ::= <empty> | VAR <varlist> *)
PROCEDURE vars(routineName:symbol);
BEGIN
    IF getTokenClass() = Var THEN
        varlist(routineName);
    ELSE
        ungetToken;
    END;
END vars;

(* <varlist> ::= <decl> | <decl> <varlist>
We can recognize the end of a varlist by seeing if the next token is an
identifier. An Id indicates the varlist continues. If it didn't we'd
see a keyword: either Begin, Procedure or Function. *)
PROCEDURE varlist(routineName:symbol);
BEGIN

```



```

    decl(routineName);
    IF peekTokenClass() = Identifier THEN
        varlist(routineName);
    END;
END varlist;

(* <decl> ::= <idlist> : <type> ;
   Declarations. All the work of putting information about the variables into
   the symbol table is done here. *)
PROCEDURE decl(routineName:symbol);
VAR tl, tokenp:tokenList;
    t, id:token;
    typeSymbol:symbol;
BEGIN
    tl := idlist();
    tokenErrorCheck(Colon, 'colon expected');
    typeSymbol := type();
    tokenErrorCheck(Semicolon, 'semicolon expected');
    tokenp := tl;
    (* Enter the variables into the symbol table. For globals, also generate
       the variables. *)
    WHILE NOT tlEmpty(tokenp) DO
        tlToken(tokenp, id);
        IF currentLexLevel() = 0 THEN
            genGlobal(enterSymbol(id.string, Global, typeSymbol));
        ELSE
            enterLocal(id.string, typeSymbol, routineName);
        END;
        tokenp := tlNext(tokenp);
    END;
    freeTokenList(tl);
END decl;

(* <idlist> ::= <id> | <id> , <idlist> *)
PROCEDURE idlist():tokenList;
VAR t: token;
BEGIN
    getTokenErrorCheck(t, Identifier, 'identifier expected');
    IF getTokenClass() <> Comma THEN (* this is the end of the idlist *)
        ungetToken;
        IF t.class = Identifier THEN
            RETURN addToTokenList(t, emptyTokenList);
        ELSE
            RETURN emptyTokenList;
        END;
    (* we saw a comma, so there's more *)
    ELIF t.class = Identifier THEN
        RETURN addToTokenList(t, idlist());
    ELSE
        RETURN idlist();
    END;
END;
END idlist;

(***) blocks and statements (***)

(* <block> ::= BEGIN <stmts> END *)
PROCEDURE block(routine:symbol):node;
VAR n:node;
BEGIN
    tokenErrorCheck(Begin, 'BEGIN expected');
    n := stmts(routine);
    tokenErrorCheck(End, '"END" expected');
    RETURN n;
END block;

(* <stmts> ::= <empty> | <stmt> ; <stmts>
   We can recognize an empty <stmts> by seeing if the next token is ELSE,
   ELIF or END. *)
PROCEDURE stmts(routine:symbol):node;
VAR n:node;
    tc:tokenClass;
BEGIN
    tc := peekTokenClass();
    IF (tc = Else) OR (tc = Elif) OR (tc = End) THEN
        RETURN emptyNode;

```

(continued)

```

ELSE
    n := stmt(routine);
    tokenErrorCheck(Semicolon, 'a semicolon must end a statement');
    RETURN makeStmtsNode(n, stmts(routine));
END;
END stmts;

(* <stmt> ::= <while> | <if> | <return> | <assign> | <call> |
    <write> | <read> *)
PROCEDURE stmt(routine:symbol):node;
VAR t:token;
BEGIN
    getToken(t);
    CASE t.class OF
        If: RETURN ifStmt(routine);
        While: RETURN whileStmt(routine);
        Return: RETURN returnStmt(routine);
        IF Symbol.equal(routine, programName) THEN
            compError("can't return from main program");
        ELSE
            RETURN makeReturnNode(routine, expr());
        END;
        Write: RETURN makeWriteNode(actuals());
        Read: RETURN makeReadNode(readActuals());
        Identifier: RETURN assignOrCallStmt(t);
    ELSE
        compError('illegal statement type');
        RETURN emptyNode;
    END;
END;
END stmt;

(* <if> ::= IF <elsif> END *)
PROCEDURE ifStmt(routine:symbol):node;
VAR n:node;
BEGIN
    n := elsif(routine);
    tokenErrorCheck(End, 'END expected');
    RETURN n;
END ifStmt;

(* <elsif> ::= <expr> THEN <stmts> <else> *)
PROCEDURE elsif(routine:symbol):node;
VAR n1, n2:node;
BEGIN
    n1 := expr();
    boolCheck(n1);
    tokenErrorCheck(Then, 'THEN expected');
    n2 := stmts(routine);
    RETURN makeIfNode(n1, n2, else(routine));
END elsif;

(* <else> ::= <empty> | ELSIF <elsif> | ELSE <stmts>
    We can tell an <else> is empty by seeing if the next token is END. *)
PROCEDURE else(routine:symbol):node;
BEGIN
    CASE getTokenClass() OF
        End: ungetToken;
            RETURN emptyNode;
        Elif: RETURN makeStmtsNode(elsif(routine), emptyNode);
        Else: RETURN stmts(routine);
    ELSE
        compError('END, ELSIF or ELSE expected');
        ungetToken;
        RETURN emptyNode;
    END;
END else;

(* <while> ::= WHILE <expr> DO <stmts> END *)
PROCEDURE whileStmt(routine:symbol):node;
VAR n:node;
BEGIN
    n := expr();
    boolCheck(n);
    tokenErrorCheck(Do, 'DO expected');
    n := makeWhileNode(n, stmts(routine));

```



```

    tokenErrorCheck(End, 'END expected');
    RETURN n;
END whileStmt;

(* <return> ::= RETURN | RETURN <expr> *)
PROCEDURE returnStmt(routine:symbol):node;
BEGIN
    IF Symbol.equal(routine, programName) THEN
        compError("can't return from main program");
    END;
    IF peekTokenClass() = Semicolon THEN
        RETURN makeReturnNode(routine, emptyNode);
    ELSE
        RETURN makeReturnNode(routine, expr());
    END;
END returnStmt;

(* We can't distinguish an assignment from a call based on the first token
of the statement, since in both cases it's an identifier. The next token,
though, will distinguish: it's an assignment sign or a left bracket
for an assignment. *)
PROCEDURE assignOrCallStmt(t:token):node;
VAR tc:tokenClass;
BEGIN
    c := peekTokenClass();
    IF (tc = Assignment) OR (tc = Lbracket) THEN
        RETURN assignStmt(t);
    ELSE
        RETURN callStmt(t);
    END;
END assignOrCallStmt;

(* <assign> ::= <idOrIndex> := <expr> *)
PROCEDURE assignStmt(varName:token):node;
VAR n:node;
BEGIN
    n := idOrIndex(findSymbol(varName.string));
    IF NOT assignable(n) THEN
        compError('cannot assign to this');
        tokenErrorCheck(Assignment, ":= expected");
        RETURN expr(); (* consume the expression anyway *)
    ELSE
        tokenErrorCheck(Assignment, ":= expected");
        RETURN makeAssignmentNode(n, expr());
    END;
END assignStmt;

(* <call> ::= <id> <actuals> *)
PROCEDURE callStmt(routineName:token):node;
VAR proc:symbol;
BEGIN
    proc := findSymbol(routineName.string);
    IF NOT Symbol.classEqual(proc, Proc) THEN
        compError('only procedures can be used in a call statement');
        RETURN actuals(); (* consume the actuals anyway *)
    ELSE
        RETURN makeCallNode(proc, actuals());
    END;
END callStmt;

(* <actuals> ::= <empty> | ( <exprlist> )
We can recognize an empty <actuals> by seeing if the next character is a
left parenthesis. *)
PROCEDURE actuals():node;
VAR n:node;
BEGIN
    IF getTokenClass() = Lparen THEN
        n := exprlist();
        tokenErrorCheck(Rparen, 'right paren expected');
        RETURN n;
    ELSE
        ungetToken;
        RETURN emptyNode;
    END;
END actuals;

```

(continued)

```
(* <exprlist> ::= <expr> | <expr> , <exprlist>
   Exprlist always returns an nList node, even if there's only one expr. *)
```

```
PROCEDURE exprlist():node;
VAR n:node;
BEGIN
  n := expr();
  IF getTokenClass() = Comma THEN
    RETURN makeExprListNode(n, exprlist());
  ELSE
    ungetToken;
    RETURN makeExprListNode(n, emptyNode);
  END;
END exprlist;
```

```
(* These two procedures are for the args to READ *)
```

```
PROCEDURE readActuals():node;
VAR n:node;
BEGIN
  IF getTokenClass() = Lparen THEN
    n := readExprlist();
    tokenErrorCheck(Rparen, 'right paren expected');
    RETURN n;
  ELSE
    ungetToken;
    RETURN emptyNode;
  END;
END readActuals;
```

```
PROCEDURE readExprlist():node;
VAR t:token;
    n:node;
BEGIN
  getTokenErrorCheck(t, Identifier, "identifier expected");
  n := idOrIndex(findSymbol(t.string));
  IF getTokenClass() = Comma THEN
    RETURN makeExprListNode(n, readExprlist());
  ELSE
    ungetToken;
    RETURN makeExprListNode(n, emptyNode);
  END;
END readExprlist;
```

```
BEGIN
END Parser.
```

```
=====
Start Routines.DEF
=====
```

```
DEFINITION MODULE Routines;
```

```
(* The part of the parser that deals with procedures and functions.
```

```
Syntax:
```

```
<routines> ::= <empty> | <proc> <routines> | <func> <routines>
<proc> ::= procedure <id> <formals> ; <vars> <block> ;
<func> ::= function <id> <formals> : <id> ; <vars> <block> ;
<formals> ::= <empty> | ( <formlist> )
<formlist> ::= <formdecl> | <formdecl> ; <formlist>
<formdecl> ::= <idlist> : <mode> <typeId>
<mode> ::= <empty> | IN | OUT | IN OUT
<typeId> ::= <id> | ARRAY OF <id>
```

```
Changes for part 3:
```

1. Modes handled.
2. Open array parameters handled.

```
*)
```

```
EXPORT QUALIFIED routines;
```


PROCEDURE routines:

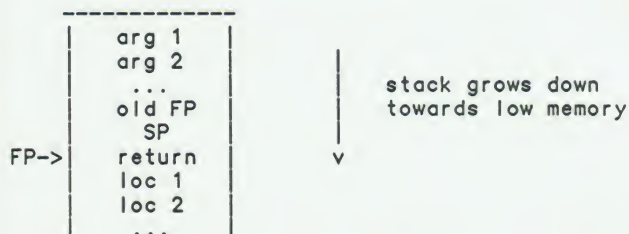
END Routines.

```
=====
Start Routines.MOD
=====
```

IMPLEMENTATION MODULE Routines:

```
(* The part of the parser that handles procedures and functions.
```

There are basically two things that have to be done: the routine declarations have to be processed to yield symbol table entries, and the code for the routine bodies has to be generated. The lists of formal parameters (arguments) and locals variables are placed in the appropriate slots in the symbol table entry for the routine. For functions, the return type of the function is put in the type slot of the symbol; for procedures, this slot is left undefined. An offset from the stack pointer is given to each local and formal. The initial offsets assume the following stack conventions:



The list of formals must be backwards to match the argument conventions. The order of the locals doesn't matter, but it's also backwards.

We generate code as if for a block, with two exceptions: at the beginning, we have to push enough words to move the stack pointer past the local storage area; while we are at it, we initialize the words to 0. At the end, we generate a return, in case the user didn't. For procedures, it is okay to return by falling off the end. For functions, something has to be returned explicitly; it is an error to fall through.

Changes for part 3:

1. Formal parameter modes handled.
2. Arrays handled. Always passed by reference; starting address passed.
3. Open array params handled. The bounds get offsets just below the starting address.

*)

```

FROM Token IMPORT token, tokenClass,
    tokenList, tIsEmpty, tIsNext, tIsToken, freeTokenList;
FROM LexAN IMPORT getToken, getTokenClass, ungetToken, tokenErrorCheck,
    getTokenErrorCheck, compError, peekTokenClass;
FROM Symbol IMPORT symbol, isType, symbolList, sIsNext, sIsSymbol, sIsEmpty,
    Class, numLocals, modeType, newArrayType;
IMPORT Symbol;
FROM SymbolTable IMPORT enterSymbol, enterFormal, beginRoutine, endRoutine,
    tUnknown, findSymbol, currentLexLevel;
FROM CodeGen IMPORT genBlock, genLocals;
FROM CodeWrite IMPORT writeInt, writeFReturn, writeReturn, writeRoutineLabel,
    writeStrings;
FROM Parser IMPORT vars, types, idlist, block;
FROM Node IMPORT node;

```

```
CONST
    initialLocalOffset = -1;      (* offsets, in words, from the FP *)
    initialFormalOffset = 3;
```

```
(* <routines> ::= <empty> | <proc> <routines> | <func> <routines> *)
PROCEDURE routines;
BEGIN
```

(continued)

```

    LOOP
        CASE getTokenClass() OF
            Procedure: proc;
            | Function: func;
            ELSE ungetToken; EXIT;
            END;
        END;
    END routines;

(* <proc> ::= procedure <id> <formals> ; <types> <vars> <routines> <block> ; *)
PROCEDURE proc;
VAR t:token;
    s:symbol;
    i:CARDINAL;
BEGIN
    getTokenErrorCheck(t, Identifier, 'procedure name expected');
    s := enterSymbol(t.string, Proc, tUnknown);
    beginRoutine(s);
    formals(s);
    tokenErrorCheck(Semicolon, 'semicolon expected');
    types(s);
    locals(s);
    routines;
    writeRoutineLabel(s);
    genLocals(s);
    genBlock(block(s));
    writeReturn(sizeFormals(s));
    writeStrings;
    tokenErrorCheck(Semicolon, 'semicolon expected');
    endRoutine(s);
END proc;

(* <func> ::= function <id> <formals>:<id>; <vars> <routines> <block>; *)
PROCEDURE func;
VAR fname, ftype:token;
    s, typeSymbol:symbol;
    i:CARDINAL;
BEGIN
    getTokenErrorCheck(fname, Identifier, 'function name expected');
    s := enterSymbol(fname.string, Func, tUnknown);
    beginRoutine(s);
    formals(s);
    tokenErrorCheck(Colon, 'colon expected');
    typeSymbol := typeName();
    IF Symbol.class(typeSymbol) <> ScalarType THEN
        compError("functions can only return scalar types");
    END;
    tokenErrorCheck(Semicolon, 'semicolon expected');
    Symbol.setType(s, typeSymbol);
    types(s);
    locals(s);
    routines;
    writeRoutineLabel(s);
    genLocals(s);
    genBlock(block(s));
    (* Here we should generate an error message in the code: value not
       returned from function. But since we have no string manipulation,
       we can't. Instead we'll return either 0 (for an Integer function) or
       false (which is also 0) for a boolean function. *)
    writeInt(0);
    writeFReturn(sizeFormals(s));
    writeStrings;
    tokenErrorCheck(Semicolon, 'semicolon expected');
    endRoutine(s);
END func;

(* <formals> ::= <empty> | ( <formlist> ) *)
PROCEDURE formals(routine:symbol);
VAR formList:symbolList;
    offset:INTEGER;
BEGIN
    IF getTokenClass() <> Lparen THEN
        ungetToken;
    ELSE
        formlist(routine);
        tokenErrorCheck(Rparen, 'right paren expected');
    END;
END formals;

```



```

(* Set the offsets of the formals *)
formList := Symbol.formals(routine);
offset := initialFormalOffset;
WHILE NOT sIsEmpty(formList) DO
    Symbol.setOffset(slSymbol(formList), offset);
    IF openArrayFormal(slSymbol(formList)) THEN
        INC(offset, 3); (* 2 extra: one for each bound *)
    ELSE
        INC(offset);
    END;
    formList := slNext(formList);
END;
END formals;

PROCEDURE openArrayFormal(s:symbol):BOOLEAN;
BEGIN
    RETURN (Symbol.class(Symbol.type(s)) = ArrayType) AND
        Symbol.open(Symbol.type(s));
END openArrayFormal;

(* <formlist> ::= <formdecl> | <formdecl> ; <formlist> *)
PROCEDURE formlist(routine:symbol);
BEGIN
    formdecl(routine);
    IF getTokenClass() = Semicolon THEN
        formlist(routine);
    ELSE
        ungetToken;
    END;
END formlist;

(* <formdecl> ::= <idlist> : <mode> <typeId> *)
PROCEDURE formdecl(routine:symbol);
VAR tl, tokenp:tokenList;
    t:token;
    typeSymbol:symbol;
    m:modeType;
BEGIN
    tl := idlist();
    tokenErrorCheck(Colon, "colon expected");
    m := formalMode();
    getToken(t);
    IF t.class = Identifier THEN
        ungetToken;
        typeSymbol := typeName();
    ELSIF t.class = Array THEN
        tokenErrorCheck(Of, "OF expected");
        typeSymbol := newArrayType(typeName(), 0, 0, TRUE,
            currentLexLevel());
        (* bounds are irrelevant; TRUE indicates open array *)
    ELSE
        compError("type name or open array parameter expected");
        ungetToken;
        typeSymbol := tUnknown;
    END;
    (* create and enter the symbols *)
    tokenp := tl;
    WHILE NOT tIsEmpty(tokenp) DO
        tlToken(tokenp, t);
        enterFormal(t.string, m, typeSymbol, routine);
        tokenp := tlNext(tokenp);
    END;
    freeTokenList(tl);
END formdecl;

(* <mode> ::= <empty> | IN | OUT | IN OUT.
Note: OUT IN is not acceptable. *)
PROCEDURE formalMode():modeType;
VAR t:token;
BEGIN
    getToken(t);
    IF t.class = In THEN
        IF getTokenClass() = Out THEN
            RETURN mInOut;
        ELSE

```

```

        ungetToken;
        RETURN mIn;
    END;
    ELIF t.class = Out THEN
        RETURN mOut;
    ELSE
        ungetToken;
        RETURN mIn;
    END;
END formalMode;

PROCEDURE locals(routine:symbol);
(* Syntactically, locals look just like globals; but we have to put them
into the locals list of the routine and give them offsets from the frame
pointer. Note that since the array handling stuff expects the starting
address to be the lowest in the array, we have to assign the offset
of arrays so that the starting address is lowest. *)
VAR locList:symbolList;
    offset:INTEGER;
    size:CARDINAL;
BEGIN
    vars(routine);
    locList := Symbol.locals(routine);
    offset := initialLocalOffset;
    (* set the offsets of the locals *)
    WHILE NOT sIsEmpty(locList) DO
        size := Symbol.size(Symbol.type(sISymbol(locList)));
        Symbol.setOffset(sISymbol(locList), offset-INTEGER(size)+1);
        DEC(offset, INTEGER(size));
        locList := sNext(locList);
    END;
END locals;

PROCEDURE typeName():symbol;
VAR t:token;
    s:symbol;
BEGIN
    getTokenErrorCheck(t, Identifier, "type name expected");
    s := findSymbol(t.string);
    IF NOT isType(s) THEN
        compError('type name expected');
        s := tUnknown;
    END;
    RETURN s;
END typeName;

PROCEDURE sizeFormals(routine:symbol):CARDINAL;
(* Returns the number of words occupied by formals. Before this was just
the number of formals, but now each open array param adds 2 to the count.
(For its bounds.) *)
VAR formList:symbolList;
    sum:CARDINAL;
    t:symbol;
BEGIN
    formList := Symbol.formals(routine);
    sum := 0;
    WHILE NOT sIsEmpty(formList) DO
        t := Symbol.type(sISymbol(formList));
        IF (Symbol.class(t) = ArrayType) AND Symbol.open(t) THEN
            INC(sum, 3);
        ELSE
            INC(sum);
        END;
        formList := sNext(formList);
    END;
    RETURN sum;
END sizeFormals;

BEGIN
END Routines.

=====
Start Symbol.DEF
=====

```


DEFINITION MODULE Symbol;

(* The symbol data structure contains all the information about symbols (like variables and routine names). Symbol lists are used for lists of formals and locals.

Changes made for part 3:

1. The word "symbol" has been removed from all routine names. Instead, the module will be imported whole (IMPORT Symbol) and the module name will serve to identify the routines (e.g. Symbol.empty). Also, SymbolClass --> Class.
2. symbolTokClass --> tokClass; setSymbolTokClass --> setTokenClass.
3. The number of symbol classes has been expanded.
4. The function tokenClassToType has been moved here from Token to avoid circular references between Token.DEF and Symbol.DEF.
5. The type mode and the function tokenClassToMode have been added.

*)

FROM Token IMPORT stringType, tokenClass;

EXPORT QUALIFIED symbol, emptySymbol, Class, class, string, type, lexLevel, offset, formals, locals, next, prev, tokClass, setFormals, setLocals, setType, setNext, setPrev, setOffset, setTokenClass, classEqual, empty, equal, new, free, numFormals, numLocals, symbolList, emptySymbolList, sIsEmpty, sIsSymbol, sIsNext, addToSymbolList, freeSymbolList, modeType, isType, mode, setMode, size, setSize, highBound, lowBound, setBounds, open, setOpen, newArrayType, anonymous, copyArrayType, setString;

TYPE

```
symbol;
symbolList;
Class = (* the different kinds of symbols *)
        (Proc, Func, ScalarType, ArrayType, Global, Local, Formal, Keyword,
         Undeclared);
modeType = (mIn, mOut, mInOut);
```

```
VAR emptySymbol:symbol;
    emptySymbolList:symbolList;

    (** Symbols **)
```

```
PROCEDURE class(s:symbol):Class;
(* Return the class of the symbol *)
```

```
PROCEDURE string(s:symbol; VAR str:stringType);
PROCEDURE setString(s:symbol; str:ARRAY OF CHAR);
(* Return or set the name of the symbol, as a string *)
```

(* Symbols are declared to be of a certain type (except procedures) *)

```
PROCEDURE type(s:symbol):symbol;
PROCEDURE setType(s, t:symbol);
```

```
PROCEDURE lexLevel(s:symbol):CARDINAL;
(* Return the lexical level at which the symbol was declared *)
```

(* Each formal and local has an offset from the frame pointer. *)

```
PROCEDURE offset(s:symbol):INTEGER;
PROCEDURE setOffset(s:symbol; o:INTEGER);
```

```
(* Formals also have a mode. *)
PROCEDURE mode(s:symbol):modeType;
PROCEDURE setMode(s:symbol; m:modeType);
```

```
(* Type objects have an associate size. *)
PROCEDURE size(s:symbol):CARDINAL;
PROCEDURE setSize(s:symbol; c:CARDINAL);
```

(* Array types have bounds, and a Boolean for open array params. *)

```
PROCEDURE lowBound(s:symbol):INTEGER;
PROCEDURE highBound(s:symbol):INTEGER;
PROCEDURE setBounds(s:symbol; low, high:INTEGER);
```

(continued)

```

PROCEDURE open(s:symbol):BOOLEAN;
PROCEDURE setOpen(s:symbol; b:BOOLEAN);

(* These are for routines. They get and set the lists of formals and locals. *)
PROCEDURE formals(s:symbol):symbolList;
PROCEDURE locals(s:symbol):symbolList;
PROCEDURE setFormals(s:symbol; sl:symbolList);
PROCEDURE setLocals(s:symbol; sl:symbolList);

(* Return the number of formals or locals in the routine. *)
PROCEDURE numFormals(s:symbol):CARDINAL;
PROCEDURE numLocals(s:symbol):CARDINAL;

(* These next two are for implementing a doubly linked list. See SymbolTable.*)
PROCEDURE next(s:symbol):symbol;
PROCEDURE prev(s:symbol):symbol;
PROCEDURE setNext(s1, s2:symbol);
PROCEDURE setPrev(s1, s2:symbol);

(* Keyword symbols have a corresponding token class. *)
PROCEDURE tokClass(s:symbol):tokenClass;
PROCEDURE setTokenClass(s:symbol; tc:tokenClass);

PROCEDURE classEqual(s:symbol; sc:Class):BOOLEAN;
(* Returns TRUE if the class of s equals sc. *)

PROCEDURE equal(s1, s2:symbol):BOOLEAN;
(* Returns TRUE if the two symbols are the same. *)

PROCEDURE empty(s:symbol):BOOLEAN;
(* Returns TRUE if the symbol is the emptySymbol. *)

PROCEDURE anonymous(s:symbol):BOOLEAN;
(* TRUE iff the array type is anonymous, i.e. unnamed. *)

PROCEDURE new(str:stringType; sc:Class; scop:CARDINAL;
              typ:symbol):symbol;
(* Creates a new symbol. *)

PROCEDURE newArrayType(baseType:symbol; lowBound, highBound:INTEGER;
                       open:BOOLEAN; lexLev:CARDINAL):symbol;
(* Creates a new array type object. *)

PROCEDURE copyArrayType(s:symbol):symbol;
(* Copies an array type object *)

PROCEDURE free(s:symbol);
(* Frees the storage associated with s. *)

PROCEDURE isType(s:symbol):BOOLEAN;
(* TRUE if the class of s is ArrayType or ScalarType. *)

    (** Symbol Lists **)

PROCEDURE slEmpty(sl:symbolList):BOOLEAN;
(* Returns TRUE if sl is the empty symbol list. *)

PROCEDURE slNext(sl:symbolList):symbolList;
(* Gets the rest of the symbol list. *)

PROCEDURE slSymbol(sl:symbolList):symbol;
(* Gets the first symbol in the list *)

PROCEDURE addToSymbolList(s:symbol; sl:symbolList):symbolList;
(* Adds s to sl at the front. Return the new symbol list. *)

PROCEDURE freeSymbolList(sl:symbolList);
(* Frees the storage associate with sl (but NOT the storage of the symbols
  in sl! *)

END Symbol.

=====
Start Symbol.MOD
=====

```


IMPLEMENTATION MODULE Symbol;

(* Symbol and symbol list data structures.

Changes made for part 3:

1. The symbolRec data structure has been reorganized.
2. Procedures have been added to access the new fields in a symbol, and to create array type objects.
3. roffset for routines: used only for the hack (in SymbolTable.beginRoutine) that assigns a unique number to non-global routines.

*)

```
FROM Token IMPORT stringType, tokenClass;
FROM Storage IMPORT ALLOCATE, DEALLOCATE;
FROM MyTerminal IMPORT fatal;
FROM SymbolTable IMPORT tUnknown;
FROM StringStuff IMPORT stringCopy;
```

TYPE

```
symbol = POINTER TO symbolRec;
symbolList = POINTER TO s1Rec;
```

symbolRec = RECORD

```
    string: stringType;
    lexLevel: CARDINAL;
    type: symbol;
    next, prev: symbol;
    CASE class: Class OF
        Proc, Func: formals, locals: symbolList;
                     roffset: INTEGER;
        |
        Local: loffset: INTEGER;
        Formal: foffset: INTEGER; mode: modeType;
        ScalarType: size: CARDINAL;
        |
        ArrayType: asize: CARDINAL; lowBound, highBound: INTEGER;
                     open: BOOLEAN;
        |
        Keyword: tokClass: tokenClass;
    END;
END;
```

s1Rec = RECORD

```
    sym: symbol;
    next: symbolList;
```

END;

Cset = SET OF Class;

(*** getting fields ***)

PROCEDURE class(s: symbol): Class;

BEGIN

```
    RETURN s^.class;
```

END class;

PROCEDURE string(s: symbol; VAR str: stringType);

BEGIN

```
    str := s^.string;
```

END string;

PROCEDURE type(s: symbol): symbol;

BEGIN

```
    RETURN s^.type;
```

END type;

PROCEDURE isType(s: symbol): BOOLEAN;

BEGIN

```
    RETURN (s^.class IN Cset{ArrayType, ScalarType}) OR (s = tUnknown);
```

END isType;

PROCEDURE lexLevel(s: symbol): CARDINAL;

BEGIN

```
    RETURN s^.lexLevel;
```

END lexLevel;

PROCEDURE offset(s: symbol): INTEGER;

(continued)

```

BEGIN
  IF s^.class = Formal THEN
    RETURN s^.foffset;
  ELSIF s^.class = Local THEN
    RETURN s^.loffset;
  ELSIF s^.class IN Cset{Proc, Func} THEN
    RETURN s^.roffset;
  ELSE
    fatal('Symbol.offset: not a Formal, Local or routine');
  END;
END offset;

PROCEDURE mode(s:symbol):modeType;
BEGIN
  IF s^.class = Formal THEN
    RETURN s^.mode;
  ELSE
    fatal('Symbol.mode: not a formal');
  END;
END mode;

PROCEDURE size(s:symbol):CARDINAL;
BEGIN
  IF s^.class = ArrayType THEN
    RETURN s^.asize;
  ELSIF s^.class = ScalarType THEN
    RETURN s^.size;
  ELSE
    fatal('Symbol.size: not a type object');
  END;
END size;

PROCEDURE highBound(s:symbol):INTEGER;
BEGIN
  IF s^.class = ArrayType THEN
    RETURN s^.highBound;
  ELSE
    fatal('Symbol.highBound: not an array type');
  END;
END highBound;

PROCEDURE lowBound(s:symbol):INTEGER;
BEGIN
  IF s^.class = ArrayType THEN
    RETURN s^.lowBound;
  ELSE
    fatal('Symbol.lowBound: not an array type');
  END;
END lowBound;

PROCEDURE open(s:symbol):BOOLEAN;
BEGIN
  IF s^.class = ArrayType THEN
    RETURN s^.open;
  ELSE
    fatal('Symbol.open: not an array type');
  END;
END open;

PROCEDURE formals(s:symbol):symbolList;
BEGIN
  IF s^.class IN Cset{Proc, Func} THEN
    RETURN s^.formals;
  ELSE
    fatal('Symbol.formals: not a proc or func');
  END;
END formals;

PROCEDURE locals(s:symbol):symbolList;
BEGIN
  IF s^.class IN Cset{Proc, Func} THEN
    RETURN s^.locals;
  ELSE
    fatal('locals: not a proc or func');
  END;
END locals;

```



```

PROCEDURE next(s:symbol):symbol;
BEGIN
  IF s = emptySymbol THEN
    fatal('Symbol.next: empty symbol given');
  ELSE
    RETURN s^.next;
  END;
END next;

PROCEDURE prev(s:symbol):symbol;
BEGIN
  RETURN s^.prev;
END prev;

PROCEDURE tokClass(s:symbol):tokenClass;
BEGIN
  IF s^.class = Keyword THEN
    RETURN s^.tokClass;
  ELSE
    fatal('Symbol.tokClass: not a keyword');
  END;
END tokClass;

      (** setting fields **)

PROCEDURE setString(s:symbol; str:ARRAY OF CHAR);
VAR i:CARDINAL;
BEGIN
  stringCopy(s^.string, str);
END setString;

PROCEDURE setFormals(s:symbol; sl:symbolList);
BEGIN
  IF s^.class IN Cset{Proc, Func} THEN
    s^.formals := sl;
  ELSE
    fatal('setFormals: not a proc or func');
  END;
END setFormals;

PROCEDURE setLocals(s:symbol; sl:symbolList);
BEGIN
  IF s^.class IN Cset{Proc, Func} THEN
    s^.locals := sl;
  ELSE
    fatal('setLocals: not a proc or func');
  END;
END setLocals;

PROCEDURE setType(s1, t:symbol);
BEGIN
  s1^.type := t;
END setType;

PROCEDURE setNext(s1, s2:symbol);
BEGIN
  s1^.next := s2;
END setNext;

PROCEDURE setPrev(s1, s2:symbol);
BEGIN
  s1^.prev := s2;
END setPrev;

PROCEDURE setOffset(s:symbol; o:INTEGER);
BEGIN
  IF s^.class = Formal THEN
    s^.foffset := o;
  ELSIF s^.class = Local THEN
    s^.loffset := o;
  ELSIF s^.class IN Cset{Proc, Func} THEN
    s^.roffset := o;
  ELSE
    fatal('Symbol.setOffset: not a formal, local, or routine');
  END;
END setOffset;

```

(continued)

```

PROCEDURE setMode(s:symbol; m:modeType);
BEGIN
  IF s^.class = Formal THEN
    s^.mode := m;
  ELSE
    fatal('Symbol.setMode: not a Formal');
  END;
END setMode;

PROCEDURE setSize(s:symbol; c:CARDINAL);
BEGIN
  IF s^.class = ScalarType THEN
    s^.size := c;
  ELSIF s^.class = ArrayType THEN
    s^.asize := c;
  ELSE
    fatal('Symbol.setSize: not a type object');
  END;
END setSize;

PROCEDURE setBounds(s:symbol; low, high:INTEGER);
BEGIN
  IF s^.class = ArrayType THEN
    s^.lowBound := low;
    s^.highBound := high;
  ELSE
    fatal('Symbol.setBounds: not an array type');
  END;
END setBounds;

PROCEDURE setOpen(s:symbol; b:BOOLEAN);
BEGIN
  IF s^.class = ArrayType THEN
    s^.open := b;
  ELSE
    fatal('Symbol.setOpen: not an array type');
  END;
END setOpen;

PROCEDURE setTokenClass(s:symbol; tc:tokenClass);
BEGIN
  IF s^.class = Keyword THEN
    s^.tokClass := tc;
  ELSE
    fatal('setTokenClass: not a keyword');
  END;
END setTokenClass;

(** other symbol procedures **)

PROCEDURE anonymous(s:symbol):BOOLEAN;
(* TRUE iff the array is anonymous *)
BEGIN
  RETURN s^.string[0] = 0C;
END anonymous;

PROCEDURE classEqual(s:symbol; sc:Class):BOOLEAN;
BEGIN
  RETURN (s^.class = Undeclared) OR (s^.class = sc);
END classEqual;

PROCEDURE equal(s1, s2:symbol):BOOLEAN;
BEGIN
  RETURN s1 = s2;
END equal;

PROCEDURE empty(s:symbol):BOOLEAN;
BEGIN
  RETURN s = emptySymbol;
END empty;

PROCEDURE new(str:stringType; sc:Class; ll:CARDINAL;
              typ:symbol):symbol;
VAR s:symbol;
BEGIN

```



```

NEW(s); (* should be: NEW(s, sc); *)
WITH s^ DO
  string := str;
  lexLevel := ll;
  type := typ;
  next := emptySymbol;
  prev := emptySymbol;
  class := sc;
  CASE class OF
    Proc, Func:
      formals := emptySymbolList;
      locals := emptySymbolList;
      | Formal: foffset := 0; mode := mIn;
      | Local: loffset := 0;
      | ArrayType: open := FALSE;
      | ScalarType: size := 1;
    ELSE (* do nothing *)
  END;
END;
RETURN s;
END new;

PROCEDURE newArrayType(baseType:symbol; lowBound, highBound:INTEGER;
  open:BOOLEAN; lexLev:CARDINAL):symbol;
VAR ato:symbol;
BEGIN
  ato := new("", ArrayType, lexLev, baseType);
  setBounds(ato, lowBound, highBound);
  setOpen(ato, open);
  setSize(ato, size(baseType) * CARDINAL(highBound-lowBound+1));
  RETURN ato;
END newArrayType;

PROCEDURE copyArrayType(s:symbol):symbol;
BEGIN
  RETURN newArrayType(type(s), lowBound(s), highBound(s), open(s),
    lexLevel(s));
END copyArrayType;

PROCEDURE free(s:symbol);
BEGIN
  DISPOSE(s); (* should be: DISPOSE(s, s^.class); *)
END free;

PROCEDURE numFormals(s:symbol):CARDINAL;
VAR formList:symbolList;
  count:CARDINAL;
BEGIN
  count := 0;
  formList := formals(s);
  WHILE NOT sIsEmpty(formList) DO
    INC(count);
    formList := sNext(formList);
  END;
  RETURN count;
END numFormals;

PROCEDURE numLocals(s:symbol):CARDINAL;
VAR locList:symbolList;
  count:CARDINAL;
BEGIN
  count := 0;
  locList := locals(s);
  WHILE NOT sIsEmpty(locList) DO
    INC(count);
    locList := sNext(locList);
  END;
  RETURN count;
END numLocals;

  (** symbolList **)

PROCEDURE sIsEmpty(sl:symbolList):BOOLEAN;
BEGIN
  RETURN sl = emptySymbolList;
END sIsEmpty;

```

(continued)

```

PROCEDURE s1Next(s1:symbolList):symbolList;
BEGIN
    RETURN s1^.next;
END s1Next;

```

```

PROCEDURE s1Symbol(s1:symbolList):symbol;
BEGIN
    RETURN s1^.sym;
END s1Symbol;

```

```

PROCEDURE addToSymbolList(s:symbol; s1:symbolList):symbolList;
VAR news1: symbolList;
BEGIN
    NEW(news1);
    news1^.sym := s;
    news1^.next := s1;
    RETURN news1;
END addToSymbolList;

```

```

PROCEDURE freeSymbolList(s1:symbolList);
BEGIN
    IF NOT s1Empty(s1) THEN
        freeSymbolList(s1Next(s1));
        DISPOSE(s1);
    END;
END freeSymbolList;

```

```

BEGIN
    emptySymbol := NIL;
    emptySymbolList := NIL;
END Symbol.

```

```

=====
Start SymbolTable.DEF
=====

```

```

DEFINITION MODULE SymbolTable;

```

```

(* The symbol table associates symbol records with names of symbols. *)

```

```

FROM Symbol IMPORT symbol, Class, modeType;
FROM Token IMPORT stringType, tokenClass;

```

```

EXPORT QUALIFIED enterSymbol, enterLocal, enterFormal, findSymbol, findKeyword,
    enterKeyword, beginRoutine, endRoutine, currentLexLevel,
    enterArrayType, tUnknown, tBoolean, tChar, tInteger, tString,
    lowFunc, highFunc;

```

```

(* These are the type objects of the built-in types. *)
VAR tUnknown, tBoolean, tChar, tInteger, tString:symbol;
    lowFunc, highFunc:symbol; (* symbols for pseudo-functions LOW & HIGH *)
PROCEDURE currentLexLevel():CARDINAL;
(* Returns the current lexical level *)

```

```

(* Enter global, local, formal, keyword symbols into the table.
   enterSymbol is the general routine and returns the entered symbol. If the
   symbol is already present, or if it is a built-in, an error is signalled.
   EnterLocal and enterFormal are used for local variables and formal
   parameters only; they take care of inserting the symbol into the list of
   locals or formals, respectively, which is associated with the routine.
   enterArrayType is for giving a name to array type objects. *)
PROCEDURE enterSymbol(VAR s:stringType; symc:Class; type:symbol):symbol;
PROCEDURE enterLocal(VAR s:stringType; type, routine:symbol);
PROCEDURE enterFormal(VAR s:stringType; mode:modeType; type, routine:symbol);
PROCEDURE enterKeyword(s:stringType; tc:tokenClass);
PROCEDURE enterArrayType(VAR s:stringType; typeObject:symbol);

```

```

PROCEDURE findSymbol(VAR s:stringType):symbol;
(* Look up the symbol in the table and return it. Return the empty symbol
   if not found. *)

```

```

PROCEDURE findKeyword(VAR s:stringType; VAR tc:tokenClass):BOOLEAN;
(* Look up the keyword in the table and put its corresponding token class
   in tc. Return FALSE if the symbol wasn't found. *)

```



```

PROCEDURE beginRoutine(rname:symbol);
(* To be called just after the routine name has been entered. Increments
lexical level. Also assigns a unique number to the routine if it isn't
global. *)

PROCEDURE endRoutine(rname:symbol);
(* Clean up the symbol table after a routine has been compiled. This includes
deleting the locals and formals from the table. *)

END SymbolTable.

=====
Start SymbolTable.MOD
=====

IMPLEMENTATION MODULE SymbolTable;

(* The symbol table for the SIMPL compiler. It is a hash table; each entry
is a symbol, possibly linked through the NEXT field to other symbols. The
list of symbols is doubly linked, to make it easy to delete from the middle.
We still have to rehash to delete from the beginning, though. This could
be gotten around by hanging a dummy record off of every hashtable entry.
*)

FROM Symbol IMPORT symbol, emptySymbol, symbolList, addToSymbolList, modeType,
    sISymbol, sINext, sIEmpty, Class, freeSymbolList, emptySymbolList;
IMPORT Symbol;
FROM Token IMPORT stringType, tokenClass;
FROM LexAn IMPORT compError;
FROM MyTerminal IMPORT fatal;
FROM StringStuff IMPORT stringEqual;

CONST symTabSize = 20;
(* This is NOT an upper limit on the number of symbols,
since we have linked lists coming off of the hashtable entries. Still,
the compiler may run faster (because the lists it searches are shorter)
if this number is increased. *)

VAR symbolTable: ARRAY[0..symTabSize-1] OF symbol;
    lexicalLevel: CARDINAL;

PROCEDURE currentLexLevel():CARDINAL;
BEGIN
    RETURN lexicalLevel;
END currentLexLevel;

PROCEDURE enterLocal(VAR s:stringType; type, routine:symbol);
VAR sym:symbol;
BEGIN
    sym := enterSymbol(s, Local, type);
    IF NOT Symbol.empty(sym) THEN
        Symbol.setLocals(routine,
            Symbol.addToSymbolList(sym, Symbol.locals(routine)));
    END;
END enterLocal;

PROCEDURE enterFormal(VAR s:stringType; mode:modeType; type, routine:symbol);
VAR sym:symbol;
BEGIN
    sym := enterSymbol(s, Formal, type);
    IF NOT Symbol.empty(sym) THEN
        Symbol.setMode(sym, mode);
        Symbol.setFormals(routine,
            Symbol.addToSymbolList(sym, Symbol.formals(routine)));
    END;
END enterFormal;

PROCEDURE enterKeyword(s:stringType; tc:tokenClass);
VAR sym:symbol;
BEGIN
    sym := enterSymbol(s, Keyword, tUnknown);
    Symbol.setTokenClass(sym, tc);
END enterKeyword;

```

(continued)

```

PROCEDURE enterArrayType(VAR s:stringType; typeObject:symbol);
(* Enter the array type object into the symbol table with the given name.
Does NOT create a new symbol. *)
BEGIN
  IF NOT isBUILTIn(s) THEN
    Symbol.setString(typeObject, s);
    typeObject (* dummy *) := insert(typeObject, hash(s));
  ELSE
    compError("cannot redefine a built-in name");
  END;
END enterArrayType;

(** symbol insertion **)

PROCEDURE enterSymbol(VAR s:stringType; symc:Symbol.Class; type:symbol):symbol;
(* This does the real work of entering a symbol. It signals an error
if a symbol is redefined, or a built-in. *)
BEGIN
  IF NOT isBUILTIn(s) THEN
    RETURN enterSym(s, symc, type);
  ELSE
    compError("can't redefine a built-in symbol");
    RETURN emptySymbol;
  END;
END enterSymbol;

PROCEDURE enterSym(VAR s:stringType; symc:Symbol.Class; type:symbol):symbol;
(* enters a symbol without doing built-in checking *)
VAR sym:symbol;
    h:CARDINAL;
BEGIN
  sym := lookup(s, FALSE, h);
  IF Symbol.empty(sym) THEN
    RETURN insert(Symbol.new(s, symc, lexicalLevel, type), h);
  ELSE
    compError('redefined symbol');
    RETURN sym;
  END;
END enterSym;

(** symbol lookup **)

PROCEDURE findSymbol(VAR s:stringType):symbol;
VAR sym:symbol;
    h: CARDINAL;
BEGIN
  sym := lookup(s, TRUE, h);
  IF Symbol.empty(sym) THEN
    compError('undefined symbol');
    RETURN insert(Symbol.new(s, Undeclared, 0, tUnknown), h);
  ELSE
    RETURN sym;
  END;
END findSymbol;

PROCEDURE findKeyword(VAR s:stringType; VAR tc:tokenClass):BOOLEAN;
(* This is used by the lexical analyzer to return the keyword's token class.
Returns true if the keyword is found; tc will then contain the token
class of the keyword. *)
VAR sym:symbol;
    h:CARDINAL;
BEGIN
  sym := lookup(s, TRUE, h);
  IF Symbol.empty(sym) OR (Symbol.class(sym) <> Keyword) THEN
    RETURN FALSE;
  ELSE
    tc := Symbol.tokClass(sym);
    RETURN TRUE;
  END;
END findKeyword;

PROCEDURE lookup(VAR s:stringType; anything:BOOLEAN; VAR h:CARDINAL):symbol;
(* Looks up the string in the symbol table. Returns the empty symbol if
the string isn't found; if it is, returns the symbol and, in h, the hash
value. anything TRUE means: "match anything".

```


This is what findSymbol uses. We match lexical level on insertion, to check for redefined symbols.

```
*)
VAR sym: symbol;
    syms: stringType;
BEGIN
    h := hash(s);
    sym := symbolTable[h];
    WHILE NOT Symbol.empty(sym) DO
        Symbol.string(sym, syms);
        IF stringEqual(syms, s) AND
            (anything OR (lexicalLevel = Symbol.lexLevel(sym))) THEN
            RETURN sym;
        END;
        sym := Symbol.next(sym);
    END;
    RETURN Symbol.emptySymbol;
END lookup;
```

```
PROCEDURE insert(s:symbol; h:CARDINAL):symbol;
(* Link the symbol into the h'th symbol table entry. The symbol is put at
the front of the list. *)
```

```
BEGIN
    Symbol.setNext(s, symbolTable[h]);
    Symbol.setPrev(s, Symbol.emptySymbol);
    symbolTable[h] := s;
    RETURN s;
END insert;
```

```
MODULE begRout; (* This needs to be a module because a variable needs
to be remembered across invocations. *)
```

```
IMPORT symbol, lexicalLevel;
IMPORT Symbol;
EXPORT beginRoutine;
```

```
VAR num:INTEGER;
```

```
PROCEDURE beginRoutine(rname:symbol);
```

```
BEGIN
    IF Symbol.lexLevel(rname) <> 0 THEN (* assign a unique number to *)
        Symbol.setOffset(rname, num); (* non-global procedures *)
        INC(num);
    END;
    INC(lexicalLevel);
END beginRoutine;
```

```
BEGIN
    num := 0;
END begRout;
```

```
PROCEDURE endRoutine(rname:symbol);
```

```
(* This is the stuff we do at the end of compiling a procedure or function.
The free's are just to reclaim storage. The remove's remove the symbols
from the symbol table, which is important if some local symbol is
shadowing a global symbol. We remove both locals and formals, but we don't
free the formals because we need them for type checking.
We also remove the routines declared at this lexical level, and free their
formals. We find these routines by searching the entire symbol table--it
would probably be better to keep a list of them.
We also remove the types declared at this level, again by exhaustive
search.
```

```
*)
BEGIN
    removeSymbolList(Symbol.locals(rname));
    freeSymbols(Symbol.locals(rname));
    Symbol.freeSymbolList(Symbol.locals(rname));
    Symbol.setLocals(rname, Symbol.emptySymbolList);
    removeSymbolList(Symbol.formals(rname));
    removeRoutinesAtThisLevel;
    DEC(lexicalLevel);
END endRoutine;
```

```
PROCEDURE removeSymbolList(symbolp:symbolList);
```

(continued)

```

BEGIN
  WHILE NOT Symbol.sIsEmpty(symbolp) DO
    removeSymbol(Symbol.s!Symbol(symbolp));
    symbolp := Symbol.s!Next(symbolp);
  END;
END removeSymbolList;

PROCEDURE removeRoutinesAtThisLevel;
(* Remove all routines defined at this lexical level. Free their formals.
   All the symbols at this lexical level will be at the beginning of their
   respective buckets in the symbol table, and they all will be routines.
   Now removes types too. *)
VAR i: CARDINAL;
    s, next: symbol;
BEGIN
  FOR i := 0 TO symTabSize-1 DO
    s := symbolTable[i];
    WHILE (NOT Symbol.empty(s)) AND (Symbol.lexLevel(s) = lexicalLevel) DO
      IF NOT Symbol.isType(s) THEN (* it's a routine *)
        freeSymbols(Symbol.formals(s));
        Symbol.freeSymbolList(Symbol.formals(s));
      END;
      (* remove this symbol from the table *)
      next := Symbol.next(s);
      symbolTable[i] := next;
      IF NOT Symbol.empty(next) THEN
        Symbol.setPrev(next, Symbol.emptySymbol);
      END;
      Symbol.free(s);
      s := next;
    END;
  END;
END removeRoutinesAtThisLevel;

PROCEDURE removeSymbol(s: symbol);
(* Splice the symbol out of the symbol table. If the symbol is at the
   beginning of the list, we have to rehash to find the right entry.
   Otherwise, just remove it from the list. *)
VAR bucket: CARDINAL;
    syms: stringType;
BEGIN
  IF Symbol.empty(Symbol.prev(s)) THEN
    Symbol.string(s, syms);
    bucket := hash(syms);
    IF NOT Symbol.equal(symbolTable[bucket], s) THEN
      fatal('removeSymbol: error');
    ELSE
      symbolTable[bucket] := Symbol.next(s);
      IF NOT Symbol.empty(Symbol.next(s)) THEN
        Symbol.setPrev(Symbol.next(s), Symbol.emptySymbol);
      END;
    END;
  ELSE
    Symbol.setNext(Symbol.prev(s), Symbol.next(s));
    IF NOT Symbol.empty(Symbol.next(s)) THEN
      Symbol.setPrev(Symbol.next(s), Symbol.prev(s));
    END;
  END;
END removeSymbol;

PROCEDURE freeSymbols(symbolp: Symbol.symbolList);
VAR nextSymbol: Symbol.symbolList;
BEGIN
  WHILE NOT Symbol.sIsEmpty(symbolp) DO
    nextSymbol := Symbol.s!Next(symbolp);
    Symbol.free(Symbol.s!Symbol(symbolp));
    symbolp := nextSymbol;
  END;
END freeSymbols;

(** low-level stuff **)

PROCEDURE hash(VAR s: stringType): CARDINAL;
(* A simple hash function: just add up the ASCII values of the characters. *)

```



```

VAR i, sum: CARDINAL;
BEGIN
  i := 0;
  sum := 0;
  WHILE s[i] <> 0C DO
    sum := sum + ORD(s[i]);
    INC(i);
  END;
  RETURN sum MOD symTabSize;
END hash;

MODULE BuiltIns;
  IMPORT stringType, stringEqual, enterSym, fatal, symbol, emptySymbol;
  IMPORT Symbol;
  EXPORT isBuiltIn, enterBuiltIn, enterBuiltInType;

  CONST maxBuiltIns = 10;

  VAR builtIns: ARRAY[1..maxBuiltIns] OF stringType;
      nBuiltIns: [0..maxBuiltIns];

  PROCEDURE isBuiltIn(VAR s: stringType): BOOLEAN;
  VAR i: CARDINAL;
  BEGIN
    FOR i := 1 TO nBuiltIns DO
      IF stringEqual(s, builtIns[i]) THEN
        RETURN TRUE;
      END;
    END;
    RETURN FALSE;
  END isBuiltIn;

  PROCEDURE enterBuiltIn(s: stringType; symc: Symbol.Class): symbol;
  BEGIN
    IF nBuiltIns = maxBuiltIns THEN
      fatal('too many built-ins');
    ELSE
      INC(nBuiltIns);
      builtIns[nBuiltIns] := s;
      RETURN enterSym(s, symc, emptySymbol);
    END;
  END enterBuiltIn;

  PROCEDURE enterBuiltInType(s: stringType): symbol;
  (* Assumes size = 1 scalar types *)
  VAR sym: symbol;
  BEGIN
    sym := enterBuiltIn(s, Symbol.ScalarType);
    Symbol.setSize(sym, 1);
    RETURN sym;
  END enterBuiltInType;

BEGIN (* module BuiltIns *)
  nBuiltIns := 0;
END BuiltIns;

PROCEDURE initSymbolTable;
VAR i: CARDINAL;
BEGIN
  FOR i := 0 TO symTabSize-1 DO
    symbolTable[i] := emptySymbol;
  END;
END initSymbolTable;

PROCEDURE enterBuiltIns;
VAR oneArg: symbolList;
BEGIN
  tInteger := enterBuiltInType("INTEGER");
  tChar := enterBuiltInType("CHAR");
  tBoolean := enterBuiltInType("BOOLEAN");
  (* tString isn't really a built-in type, nor is STRING a reserved word.
   So just create a symbol for tString. Do be careful about
   the symbol class and the basetype: some might use that info in
   type-checking (e.g. writeCheck). Strings are arrays of CHAR. *)
  tString := Symbol.new("_tString", ArrayType, lexicalLevel, tChar);

```

(continued)

```

(* tUnknown is a total dummy, but it shouldn't be equal to the empty
   symbol nonetheless. *)
tUnknown := Symbol.new("_tUnknown", ScalarType, lexicalLevel, emptySymbol);
(* LOW and HIGH built-in functions can be treated as ordinary
   functions by everything except the code generator. Give them a dummy
   formal for type-checking purposes. The formal will correctly default to
   mode IN. *)
oneArg := addToSymbolList(Symbol.new("_dummy", Formal, lexicalLevel,
                                     tUnknown), emptySymbolList);
lowFunc := enterBuiltin("LOW", Func);
Symbol.setFormals(lowFunc, oneArg);
Symbol.setType(lowFunc, tInteger);
highFunc := enterBuiltin("HIGH", Func);
Symbol.setFormals(highFunc, oneArg);
Symbol.setType(highFunc, tInteger);
END enterBuiltIns;

```

```

BEGIN
  lexicalLevel := 0;
  initSymbolTable;
  enterBuiltIns;
END SymbolTable.

```

```

=====
Start Token.DEF
=====

```

```

DEFINITION MODULE Token;

```

```

(* Tokens are what the lexical analyzer returns to the parser. Keywords are
   distinct tokens, as are the special characters like parens, colon, etc.
   TokenLists are lists of tokens; they are used in the "varlist"
   procedure of the parser.

```

```

Changes made for part 3:

```

1. New tokens have been added for the new constructs.
2. The function tokenClassToType has been moved to Symbol to avoid circular references between Token.DEF and Symbol.DEF.
3. The function isType has been renamed isBuiltinType.

```

*)

```

```

EXPORT QUALIFIED token, tokenClass, stringType, stringlen, isRelation,
  tokenList, emptyTokenList, tIToken, tINext, addToTokenList,
  tIEmpty, freeTokenList;

```

```

CONST stringlen = 80;

```

```

TYPE

```

```

  tokenClass = (And, Array, Assignment, Begin, Character,
    Colon, Comma, Divide, Do, DotDot, Else, Elif, End, EndOfInput,
    Equal, False, Function, Greater, GreaterEqual, Identifier, If,
    In, Int, Lbracket, Less, LessEqual, Lparen, Minus, Not,
    NotEqual, Of, Or, Out, Period, Plus, Procedure,
    Program, Rbracket, Read, Return, Rparen, Semicolon, String, Then,
    Times, True, Type, UMinus, Var, While, Write);

```

```

  stringType = ARRAY[0..stringlen] OF CHAR;

```

```

  token = RECORD
    CASE class:tokenClass OF
      Identifier, String: string: stringType;
      | Int: integer: INTEGER;
      | Character: ch: CHAR;
    END;
  END;

```

```

  tokenList;

```

```

VAR emptyTokenList: tokenList;

```

```

PROCEDURE isRelation(tc:tokenClass):BOOLEAN;

```

```

(* Returns TRUE if tc is a relational operator (Equal, Greater, etc.) *)

```



```

PROCEDURE t1Token(tl:tokenList; VAR t:token);
(* Gets the first token in the token list. *)

PROCEDURE t1Next(tl:tokenList):tokenList;
(* Gets the rest of the token list. *)

PROCEDURE addToTokenList(VAR t:token; tl:tokenList):tokenList;
(* Add a token to the beginning of the token list. *)

PROCEDURE freeTokenList(tl:tokenList);
(* Free the storage used by the token list. *)

PROCEDURE t1Empty(tl:tokenList):BOOLEAN;
(* Return TRUE if the token list is empty. *)

END Token.

=====
Start Token.MOD
=====

IMPLEMENTATION MODULE Token;

(* Tokens and token lists for the SIMPL compiler. *)

FROM Storage IMPORT ALLOCATE, DEALLOCATE;
FROM Terminal IMPORT WriteString;

TYPE tokenList = POINTER TO tokenListRec;  (* token lists are linked lists *)
tokenListRec = RECORD
    tok: token;
    next: tokenList;
END;

PROCEDURE isRelation(tc:tokenClass):BOOLEAN;
BEGIN
    RETURN (tc = Equal) OR (tc = NotEqual) OR (tc = Greater) OR
           (tc = GreaterEqual) OR (tc = Less) OR (tc = LessEqual);
END isRelation;

PROCEDURE t1Token(tl:tokenList; VAR t:token);
BEGIN
    IF t1Empty(tl) THEN
        WriteString("t1Token: empty tokenList");
    ELSE
        t := tl^.tok;
    END;
END t1Token;

PROCEDURE t1Next(tl:tokenList):tokenList;
BEGIN
    RETURN tl^.next;
END t1Next;

PROCEDURE addToTokenList(VAR t:token; tl:tokenList):tokenList;
(* Create a token list record for the new token and splice it on to the
   front of the token list. Return ( a pointer to) the new record. *)
VAR newtl: tokenList;
BEGIN
    NEW(newtl);
    newtl^.tok := t;
    newtl^.next := tl;
    RETURN newtl;
END addToTokenList;

PROCEDURE freeTokenList(tl:tokenList);
BEGIN
    IF NOT t1Empty(tl) THEN
        freeTokenList(t1Next(tl));
        DISPOSE(tl);
    END;
END freeTokenList;

PROCEDURE t1Empty(tl:tokenList):BOOLEAN;
BEGIN

```

(continued)

February

```
RETURN t1 = emptyTokenList;  
END t1Empty;
```

```
BEGIN  
    emptyTokenList := NIL;  
END Token.
```

```
=====  
Start TypeChecker.DEF  
=====
```

```
DEFINITION MODULE TypeChecker;
```

```
(* Handles the actual type-checking of SIMPL expressions and statements. *)
```

```
FROM Token IMPORT tokenClass;  
FROM Node IMPORT node;  
FROM Symbol IMPORT symbol, modeType;
```

```
EXPORT QUALIFIED typeCompatible, opAppropriate, callCheck, readCheck,  
    writeCheck, boolCheck, assignCheck, binopCheck, unopCheck,  
    returnCheck, assignable, hasMode, typeIdentical,  
    indexCheck, baseType;
```

```
PROCEDURE typeCompatible(t1, t2:symbol):BOOLEAN;  
(* Returns TRUE if t1 and t2 are compatible types. In order to avoid  
cascades of error messages, if one or both of the types is tUnknown,  
it still returns TRUE. *)
```

```
PROCEDURE typeIdentical(t1, t2:symbol):BOOLEAN;  
(* Returns TRUE iff t1 and t2 are the SAME type. Again, tUnknown is allowed  
to be the same as anything. *)
```

```
PROCEDURE opAppropriate(op:tokenClass; arg:node):BOOLEAN;  
(* Returns TRUE if the type of the argument can be handled by the operator *)
```

```
PROCEDURE callCheck(routine:symbol; args:node);  
(* Checks the procedure or function call for right number and types of args. *)
```

```
PROCEDURE readCheck(actuals:node);  
(* Checks the call to the READ built-in procedure. *)
```

```
PROCEDURE writeCheck(actuals:node);  
(* Checks the call to the WRITE built-in procedure. *)
```

```
PROCEDURE boolCheck(n:node);
```

```
PROCEDURE assignCheck(var, expr:node);
```

```
PROCEDURE returnCheck(routine:symbol; expr:node):BOOLEAN;
```

```
PROCEDURE binopCheck(op:tokenClass; leftarg, rightarg:node):BOOLEAN;
```

```
PROCEDURE unopCheck(op:tokenClass; arg:node):BOOLEAN;
```

```
PROCEDURE assignable(n:node):BOOLEAN;  
(* True if s can be assigned to. *)
```

```
PROCEDURE hasMode(n:node; m:modeType):BOOLEAN;  
(* A node has a particular mode iff:  
1. The node is an index, the array being indexed is a formal and  
the formal has mode m; OR  
2. The node is a symbol, the symbol is a formal and it has mode m.  
)
```

```
PROCEDURE indexCheck(index:node);  
(* Checks for compatibility with INTEGER, not of mode OUT *)
```

```
PROCEDURE baseType(typeObject:symbol):symbol;  
(* Follows the type field of a symbol until it hits an array type or  
a built-in scalar type. *)
```

```
END TypeChecker.
```

```
=====
```


Start TypeChecker.MOD

=====

IMPLEMENTATION MODULE TypeChecker;

(* Handles type-checking of SIMPL expressions. *)

```
FROM Node IMPORT node, nodeType, nodeFirst, nodeRest, nodeEmpty, nodeClass,
  NodeClass, nodeSymbol, nodeString, nodeArray, nodeIndex;
FROM Token IMPORT tokenClass, stringType;
FROM Symbol IMPORT symbol, symbolList, s1Next, s1Symbol, s1Empty, Class,
  numFormals, emptySymbol, modeType;
IMPORT Symbol;
FROM SymbolTable IMPORT tUnknown, tInteger, tChar, tBoolean, tString,
  lowFunc, highFunc;
FROM MyTerminal IMPORT fatal;
FROM LexAn IMPORT compError;
FROM StringStuff IMPORT stringLen;
```

```
TYPE Cset = SET OF Class;
   Nset = SET OF NodeClass;
```

PROCEDURE opAppropriate(op:tokenClass; arg:node):BOOLEAN;

```
BEGIN
  CASE op OF
    Plus, Minus, UMinus, Times, Divide:
      RETURN typeCompatible(nodeType(arg), tInteger);
    | Greater, GreaterEqual, Less, LessEqual:
      RETURN typeCompatible(nodeType(arg), tInteger) OR
        typeCompatible(nodeType(arg), tChar);
    | And, Or, Not:
      RETURN typeCompatible(nodeType(arg), tBoolean);
    | Equal, NotEqual:
      RETURN typeCompatible(nodeType(arg), tInteger) OR
        typeCompatible(nodeType(arg), tChar) OR
        typeCompatible(nodeType(arg), tBoolean);
  ELSE
    fatal("opAppropriate: unknown op type");
  END;
END opAppropriate;
```

PROCEDURE typeCompatible(t1, t2:symbol):BOOLEAN;
(* Two types are compatible if they have the same base type. *)

```
BEGIN
  RETURN typeIdentical(baseType(t1), baseType(t2));
END typeCompatible;
```

PROCEDURE typeIdentical(t1, t2:symbol):BOOLEAN;
(* Two types are identical iff they are the SAME type object, or if one is tUnknown. *)

```
BEGIN
  RETURN Symbol.equal(t1, tUnknown) OR
    Symbol.equal(t2, tUnknown) OR
    Symbol.equal(t1, t2);
END typeIdentical;
```

PROCEDURE baseType(typeObject:symbol):symbol;
(* Compute the base type of typeObject. *)

```
BEGIN
  IF Symbol.empty(typeObject) THEN
    RETURN typeObject;
  END;
  WHILE NOT (Symbol.empty(Symbol.type(typeObject)) OR
    (Symbol.class(typeObject) = ArrayType)) DO
    typeObject := Symbol.type(typeObject);
  END;
  RETURN typeObject;
END baseType;
```

PROCEDURE callCheck(routine:symbol; args:node);
(* Tricky because formals are stored backwards in symbol, but forwards
in the call to the routine. We do nothing if the symbol is not a procedure
or function; that check is handled in the parser.
For part 3: special check for lowFunc and highFunc. *)

(continued)

```

VAR nFormals, nActuals: CARDINAL;
    dummy: node;
BEGIN
    IF Symbol.class(routine) IN Cset{Proc, Func} THEN
        nFormals := numFormals(routine);
        nActuals := numActuals(args);
        IF nActuals < nFormals THEN
            compError('too few arguments to routine');
        ELSIF nActuals > nFormals THEN
            compError('too many arguments to routine');
        END;
        dummy := argsMatch(Symbol.formals(routine), args,
            Symbol.equal(routine, lowFunc) OR Symbol.equal(routine, highFunc));
    END;
END calCheck;

PROCEDURE argsMatch(flist: symbolList; alist: node; lowOrHigh: BOOLEAN): node;
(* This procedure matches two lists, one of which is backwards. It does
   it by recursing down one list all the way, then iterating down the other
   list while unrecursing. *)
BEGIN
    IF sIsEmpty(flist) THEN
        RETURN alist;
    ELSE
        alist := argsMatch(sNext(flist), alist, lowOrHigh);
        IF nodeEmpty(alist) THEN
            RETURN alist;
        ELSE
            argCheck(s1Symbol(flist), nodeFirst(alist), lowOrHigh);
            RETURN nodeRest(alist);
        END;
    END;
END argsMatch;

PROCEDURE argCheck(formal: symbol; actual: node; lowOrHigh: BOOLEAN);
(* An argument matches a formal if:
   The modes are compatible (see modeCompatible, below) AND
   1. The types are IDENTICAL (not compatible) OR
   2. The formal has an open array param as a type, and the
      actual is an array of the identical base type (incl. string) OR
   3. The formal is an array of CHAR and the actual is a string
      constant of size <= the array.
   For LOW and HIGH, all that is required is that the arg be an array.
   *)
VAR ftype, atype: symbol;
BEGIN
    IF NOT nodeEmpty(actual) THEN
        IF lowOrHigh THEN
            IF NOT Symbol.classEqual(baseType(nodeType(actual)), ArrayType) THEN
                compError("LOW and HIGH take only arrays");
            END;
        ELSIF modeCompatible(formal, actual) THEN
            ftype := Symbol.type(formal);
            atype := nodeType(actual);
            IF NOT (typeIdentical(ftype, atype) OR
                openArray(ftype, atype) OR
                stringConst(ftype, actual)) THEN
                compError('type of formal does not match type of actual');
            END;
        ELSE
            compError("mode incompatibility");
        END;
    END;
END argCheck;

PROCEDURE modeCompatible(formal: symbol; actual: node): BOOLEAN;
(* TRUE iff:
   1. formal has mode IN and actual does not have mode OUT; OR
   2. formal has mode OUT and actual
      2a. is a variable or index; and
      2b. does not have mode IN; OR
   3. formal has mode IN OUT and actual
      3a. is a variable or index; and
      3b. does not have modes IN or OUT *)
BEGIN
    IF Symbol.mode(formal) = mIn THEN
        RETURN NOT hasMode(actual, mOut);
    END;

```



```

ELSIF NOT (nodeClass(actual) IN Nset{nSymbol, nIndex}) THEN
  RETURN FALSE;
ELSIF Symbol.mode(formal) = mOut THEN
  RETURN NOT hasMode(actual, mIn);
ELSIF Symbol.mode(formal) = mInOut THEN
  RETURN (NOT hasMode(actual, mIn)) AND (NOT hasMode(actual, mOut));
ELSE
  fatal('modeCompatible: unknown mode');
END;
END modeCompatible;

PROCEDURE hasMode(n:node; m:modeType):BOOLEAN;
(* A node has a particular mode iff:
  1. The node is an index and the array being indexed has mode m; OR
  2. The node is a symbol, the symbol is a formal and it has mode m.
*)
BEGIN
  IF nodeClass(n) = nIndex THEN
    RETURN hasMode(nodeArray(n), m);
  ELSE
    RETURN (nodeClass(n) = nSymbol) AND
      (Symbol.class(nodeSymbol(n)) = Formal) AND
      (Symbol.mode(nodeSymbol(n)) = m);
  END;
END hasMode;

PROCEDURE openArray(ftype, atype:symbol):BOOLEAN;
(* TRUE iff ftype is an open array and atype is an array of the identical
  base type. *)
BEGIN
  RETURN (Symbol.class(ftype) = ArrayType) AND
    Symbol.open(ftype) AND
    (Symbol.class(atype) = ArrayType) AND
    typeIdentical(Symbol.type(ftype), Symbol.type(atype));
END openArray;

PROCEDURE stringConst(ftype:symbol; actual:node):BOOLEAN;
(* TRUE iff ftype is an array of char (possibly open) and actual is tString,
  and the string const is shorter than the array. *)
VAR s:stringType;
BEGIN
  IF (Symbol.class(ftype) = ArrayType) AND
    Symbol.equal(Symbol.type(ftype), tChar) AND
    Symbol.equal(nodeType(actual), tString) THEN
    nodeString(actual, s);
    RETURN Symbol.open(ftype) OR (stringLen(s) <= Symbol.size(ftype));
  ELSE
    RETURN FALSE;
  END;
END stringConst;

PROCEDURE numActuals(actuals:node):CARDINAL;
VAR count:CARDINAL;
BEGIN
  count := 0;
  WHILE NOT nodeEmpty(actuals) DO
    INC(count);
    actuals := nodeRest(actuals);
  END;
  RETURN count;
END numActuals;

PROCEDURE readCheck(actuals:node);
VAR arg:node;
BEGIN
  IF nodeEmpty(actuals) THEN
    compError('READ requires an argument');
  ELSE
    REPEAT
      arg := nodeFirst(actuals);
      IF NOT nodeEmpty(arg) THEN
        IF NOT assignable(arg) THEN
          compError('READ must be able to assign to its arguments');
        END;
      END;
    UNTIL nodeEmpty(arg);
  END;
END;

```

(continued)

```

        IF NOT charOrInt(nodeType(arg)) THEN
            compError('READ can only read integers or characters');
        END;
    END;
    actuals := nodeRest(actuals);
    UNTIL nodeEmpty(actuals);
END;
END readCheck;

PROCEDURE writeCheck(actuals:node);
VAR arg:node;
BEGIN
    IF nodeEmpty(actuals) THEN
        compError('WRITE requires an argument');
    ELSE
        REPEAT
            arg := nodeFirst(actuals);
            IF NOT nodeEmpty(arg) THEN
                IF NOT charOrInt(nodeType(arg)) THEN
                    compError('WRITE can only write integers or characters');
                END;
            END;
            actuals := nodeRest(actuals);
        UNTIL nodeEmpty(actuals);
    END;
END writeCheck;

PROCEDURE binopCheck(op:tokenClass; leftarg, rightarg:node):BOOLEAN;
BEGIN
    IF NOT opAppropriate(op, leftarg) THEN
        compError('inappropriate arg type: left arg');
        RETURN FALSE;
    END;
    IF NOT opAppropriate(op, rightarg) THEN
        compError('inappropriate arg type: right arg');
        RETURN FALSE;
    END;
    (* The two operands must be THE SAME, not just compatible. *)
    IF NOT typeIdentical(nodeType(leftarg), nodeType(rightarg)) THEN
        compError('argument types not identical');
        RETURN FALSE;
    ELSE
        RETURN TRUE;
    END;
END binopCheck;

PROCEDURE unopCheck(op:tokenClass; arg:node):BOOLEAN;
BEGIN
    IF NOT opAppropriate(op, arg) THEN
        compError('inappropriate arg type');
        RETURN FALSE;
    ELSE
        RETURN TRUE;
    END;
END unopCheck;

PROCEDURE assignCheck(var, expr:node);
(* For assignment, the types must be identical, unless one is an open
   array, in which case only the base types need be identical. If the
   expr is a string constant, the var need only be an array of char. *)
VAR t1,t2:symbol;
BEGIN
    t1 := nodeType(var);
    t2 := nodeType(expr);
    IF NOT (openArray(t1, t2) OR
            openArray(t2, t1) OR
            typeIdentical(t1, t2) OR
            stringConst(t1, expr)) THEN
        compError('types not assignment-compatible');
    END;
END assignCheck;

PROCEDURE boolCheck(n:node);
BEGIN
    IF NOT typeCompatible(nodeType(n), tBoolean) THEN
        compError('Boolean expression expected');
    END;
END;

```



```

END boolCheck;

PROCEDURE returnCheck(routine:symbol; expr:node):BOOLEAN;
BEGIN
  IF (NOT nodeEmpty(expr)) AND (NOT Symbol.classEqual(routine, Func)) THEN
    compError('only functions can return values');
    RETURN FALSE;
  ELSIF nodeEmpty(expr) AND Symbol.classEqual(routine, Func) THEN
    compError('function must return a value');
    RETURN FALSE;
  ELSIF (NOT nodeEmpty(expr)) AND
    (NOT typeIdentical(Symbol.type(routine), nodeType(expr))) THEN
    compError('return type not identical to function type');
    RETURN FALSE;
  ELSE
    RETURN TRUE;
  END;
END returnCheck;

PROCEDURE assignable(n:node):BOOLEAN;
(* Something can be assigned to if: it is a variable (including array
   indices), and it does not have mode IN. *)
BEGIN
  RETURN variable(n) AND (NOT hasMode(n, mIn));
END assignable;

PROCEDURE indexCheck(index:node);
BEGIN
  IF NOT nodeEmpty(index) THEN
    IF hasMode(index, mOut) THEN
      compError("can't use an OUT formal to index an array");
    ELSIF NOT typeCompatible(nodeType(index), tInteger) THEN
      compError("array index must be compatible with type INTEGER");
    END;
  END;
END indexCheck;

PROCEDURE variable(n:node):BOOLEAN;
(* TRUE iff n is a symbol of class Global, Local, or Formal; or
   if n is an index. *)
BEGIN
  RETURN (NOT nodeEmpty(n)) AND
    ((nodeClass(n) = nIndex) OR
     ((nodeClass(n) = nSymbol) AND
      (Symbol.class(nodeSymbol(n)) IN Cset{Global, Local, Formal})));
END variable;

PROCEDURE charOrInt(t:symbol):BOOLEAN;
(* TRUE iff t is a type object compatible with CHAR or INTEGER *)
BEGIN
  RETURN typeCompatible(t, tChar) OR typeCompatible(t, tInteger);
END charOrInt;

BEGIN
  END TypeChecker.

```


March

mscreen.mod

TEXT
"Modula-2 System for Z80 CP/M." See mfilecpy.mod.

MODULE MSCREEN;

FROM TERM1 IMPORT (* non-standard terminal module *)
Write, WriteCard, WriteString, WriteLn;

FROM ASCII IMPORT
bel, sub;

VAR
i : CARDINAL;

BEGIN

Write (sub); (* Clear Console Screen *)
Write (bel);

FOR i := 1 TO 100 DO
WriteCard (i, 3);
WriteString (' The quick brown fox jumped over the lazy dogs back. ');
WriteString ('1234567890'); WriteLn;
END;

Write (bel);
END MSCREEN.

msieve.mod

TEXT
"Modula-2 System for Z80 CP/M." See mfilecpy.mod.

MODULE MSIEVE;

FROM TERM1 IMPORT
WriteString, WriteCard, WriteLn;

CONST
Size = 8190; (* size of array *)
Iterations = 10; (* minimum 1 *)

VAR
count, i, iter, k, prime : CARDINAL;
flags : ARRAY [0..Size] OF BOOLEAN;

BEGIN

WriteString ('10 Iterations'); WriteLn;
FOR iter := 1 TO Iterations DO
count := 0;

FOR i := 0 TO Size DO
flags[i] := TRUE;
END;

FOR i := 0 TO Size DO
IF flags[i] THEN
prime := i + i + 3;
k := i + prime;
WHILE k <= Size DO
flags[k] := FALSE;
INC (k, prime);
END;
INC (count);
END;
END; (* FOR *)
END; (* FOR *)

WriteString ('There were ');
WriteCard (count, 0); WriteString
(' primes. ');
END MSIEVE.

(continued)

mtimef.mod

TEXT
 "Modula-2 System for Z80 CP/M." See mfilecpy.mod.

```

MODULE MTIMEF;  (* times floating point
                  operations *)

FROM Terminal IMPORT
  Write, WriteLn, WriteString;

FROM MathLib IMPORT
  sin, cos, arctan, ln, exp;

FROM ASCII IMPORT
  bel;

VAR
  x, y, z : REAL;
  i : CARDINAL;

PROCEDURE Delay (x : CARDINAL);
  (* variable delay, x in milliseconds *)

  VAR
    i, j : CARDINAL;

  BEGIN
    FOR i := 1 TO x DO
      FOR j := 1 TO 18 DO
        END;
      END;
    END Delay;

BEGIN
  x := 12.5;  y := 0.5;

  WriteString ('Blank');  WriteLn;
  Delay (500);
  Write (bel);
  i := 1;
  REPEAT
    z := y;
    i := i + 1;
  UNTIL i = 10000;
  Write (bel);
  Delay (4000);

  WriteString ('Addition');  WriteLn;
  Delay (500);
  Write (bel);
  i := 1;
  REPEAT
    z := x + y;
    i := i + 1;
  UNTIL i = 10000;
  Write (bel);
  Delay (4000);

  WriteString ('Subtraction');  WriteLn;
  Delay (500);
  Write (bel);
  i := 1;
  REPEAT
    z := x - y;
    i := i + 1;
  UNTIL i = 10000;
  Write (bel);
  Delay (4000);

```

```

  WriteString ('Multiplication');  WriteLn;
  Delay (500);
  Write (bel);
  i := 1;
  REPEAT
    z := x * y;
    i := i + 1;
  UNTIL i = 10000;
  Write (bel);
  Delay (4000);

  WriteString ('Division');  WriteLn;
  Delay (500);
  Write (bel);
  i := 1;
  REPEAT
    z := x / y;
    i := i + 1;
  UNTIL i = 10000;
  Write (bel);
  Delay (4000);

  WriteString ('Sine');  WriteLn;
  Delay (500);
  Write (bel);
  i := 1;
  REPEAT
    z := sin (y);
    i := i + 1;
  UNTIL i = 10000;
  Write (bel);
  Delay (4000);

  WriteString ('Cosine');  WriteLn;
  Delay (500);
  Write (bel);
  i := 1;
  REPEAT
    z := cos (y);
    i := i + 1;
  UNTIL i = 10000;
  Write (bel);
  Delay (4000);

  WriteString ('Arctangent');  WriteLn;
  Delay (500);
  Write (bel);
  i := 1;
  REPEAT
    z := arctan (x);
    i := i + 1;
  UNTIL i = 10000;
  Write (bel);
  Delay (4000);

  WriteString ('Natural Log');  WriteLn;
  Delay (500);
  Write (bel);
  i := 1;
  REPEAT
    z := ln (x);
    i := i + 1;
  UNTIL i = 10000;
  Write (bel);
  Delay (4000);

```



```

WriteString ('Natural Antilog'); WriteLn;
Delay (500);
Write (bel);
i := 1;
REPEAT
  z := exp (y);

```

```

  i := i + 1;
UNTIL i = 1000;
Write (bel);
Delay (4000);
END MTIMEF.

```

mtimei.mod

TEXT

"Modula-2 System for Z80 CP/M." See mfilecpy.mod.

MODULE MTIMEI;

FROM Terminal IMPORT

(* standard module as defined by Wirth *)
Write, WriteLn, WriteString;

FROM ASCII IMPORT
bel;

VAR
x, y, z : INTEGER;
i : CARDINAL;

PROCEDURE Delay (x : CARDINAL);
(* variable Delay, x in milliseconds *)

VAR
i, j : CARDINAL;

BEGIN
FOR i := 1 TO x DO
FOR j := 1 TO 18 DO
END;
END;
END Delay;

BEGIN

x := 11; y := 2;

WriteString ('Blank'); WriteLn;
Delay (500);
Write (bel);
i := 1;
REPEAT
z := y;
i := i + 1;
UNTIL i = 1000;
Write (bel);
Delay (4000);

WriteString ('Addition'); WriteLn;
Delay (500);
Write (bel);
i := 1;
REPEAT
z := x + y;
i := i + 1;

UNTIL i = 1000;
Write (bel);
Delay (4000);

WriteString ('Subtraction'); WriteLn;
Delay (500);
Write (bel);
i := 1;
REPEAT

z := x - y;
i := i + 1;
UNTIL i = 1000;
Write (bel);
Delay (4000);

WriteString ('Multiplication'); WriteLn;
Delay (500);
Write (bel);
i := 1;
REPEAT

z := x * y;
i := i + 1;
UNTIL i = 1000;
Write (bel);
Delay (4000);

WriteString ('Division'); WriteLn;
Delay (500);
Write (bel);
i := 1;
REPEAT

z := x DIV y;
i := i + 1;
UNTIL i = 1000;
Write (bel);
Delay (4000);

WriteString ('Modulus'); WriteLn;
Delay (500);
Write (bel);
i := 1;
REPEAT

z := x MOD y;
i := i + 1;
UNTIL i = 1000;
Write (bel);
Delay (4000);

WriteString ('Good-bye...'); WriteLn;
END MTIMEI.

(continued)

mfilecpy.mod

TEXT

Software Review: "Modula-2 System for Z80 CP/M," Brian R. Anderson.
 March, page 225. Also download mscreen.mod, msieve.mod, mtimef.mod, and
 mtimei.mod.

MODULE MFILECPY;

```
FROM SeqIO IMPORT
  FILE, FileState, Open, Create, Close, Read, Write, EOF;
```

```
FROM Terminal IMPORT
  ReadString, WriteLn, WriteString;
```

```
FROM Strings IMPORT
  STRING;
```

```
VAR
  inFILE, outFILE : FILE;
  name, BAKname : STRING;  (* file names *)
  c : CHAR;
```

```
PROCEDURE MakeBAK (in : STRING; VAR out : STRING; tag : STRING);
```

```
VAR
  i, j : CARDINAL;
```

```
BEGIN
  i := 0;
  WHILE (in[i] # 0C) AND (in[i] # '.') DO
    out[i] := in[i];
    INC (i);
  END;

  j := 0;
  WHILE tag[j] # 0C DO
    out[i] := tag[j];
    INC (i); INC (j);
  END;

  out[i] := 0C;  (* add NULL terminator *)
END MakeBAK;
```

BEGIN

```
WriteString ('File Backup Utility');  WriteLn;  WriteLn;
WriteString ('Enter filename: ');
ReadString (name);  WriteLn;
MakeBAK (name, BAKname, '.bak');
```

```
IF Create (outFILE, BAKname) = FileOK THEN
```

```
  IF Open (inFILE, name) = FileOK THEN
```

```
    WHILE NOT EOF (inFILE) DO
```

```
      Read (inFILE, c);
```

```
      Write (outFILE, c);
```

```
    END;
```

```
  IF (Close (inFILE) <> FileOK) OR (Close (outFILE) <> FileOK) THEN
```

```
    WriteString ('Error closing files...');  WriteLn;
```

```
  ELSE
```

```
    WriteString (BAKname);  WriteString (' completed. ');  WriteLn;
```

```
  END;
```

```
ELSE
```

```
  IF Close (outFILE) <> FileOK THEN
```

```
    (* do nothing *)
```

```
  END;
```



```

        WriteString ('Error creating new file...'); WriteLn;
    END;
ELSE
    WriteString ('Error opening file...'); WriteLn;
END;
END MFILECPY.

```

listing3.386

```

TEXT
Circuit Cellar: "Real-Time Clocks: A View Toward the
Future." See listing1.386 for details.

```

<pre> 10 MTOP = MTOP - 30 (RESET MTOP POINTER) 20 DBY(18H)=040H (ASSUME SMARTWATCH AT 4000H) 30 CALL 6000H (INITIALIZE THE SYSTEM) 40 REM NOW SET SMARTWATCH TIME 50 FOR X=MTOP+24 TO MTOP+17 STEP -1 60 READ C 70 XBY(X)=C </pre>	<pre> 80 NEXT X 85 REM ZZ/01/85 14:25:00.00 90 DATA 85,11,01,05,14,25,00,00 100 CALL 6003H (WRITE THE VALUES) 110 CALL 6006H (READ THE VALUES) 120 PRINT MTOP+18 (SECONDS COUNTER) 130 GOTO 110 </pre>
--	--

listing1.386

```

TEXT
Circuit Cellar: "Real-Time Clocks: A View Toward the Future," Steve Ciarcia.
March, page 112. Also download listing2.386, and listing3.386.

```

```

100 DIM N(200) : DIM M(200)
110 REM
120 REM REV 1.5 11/8/85
130 REM 5832 REAL TIME CLOCK FOR BCC-52 I/O PORT
140 REM
150 P1=51200 : P2=51201 : P3=51202 : P4=51203
155 REM SET 8255 PORT A AS INPUT AND B&C AS OUTPUT
160 XBY(P4)=90H
170 REM PORT B IS ADDRESS AND PORT C IS CONTROL BUS
180 PRINT "ENTER 0 TO SET TIME OR 1 TO READ TIME", : INPUT A
190 ON A GOSUB 350,220
200 GOTO 180
210 GOTO 145
220 REM READ 13 5832 REGISTERS
230 XBY(P3)=20H : REM SET READ MODE
240 FOR A=0 TO 12
250 XBY(P2)=A : N(A)=XBY(P1)
260 NEXT A
270 REM DISPLAY CONTENTS
280 PRINT "DATE ",
290 PRINT N(10)*10+N(9),"/",N(8)*10+N(7),"/",N(12)*10+N(11)
300 PRINT "TIME ",
310 IF N(5)>=8 THEN N(5)=N(5)-8
320 PRINT (N(5)*10+N(4))," : ",(N(3)*10+N(2))," : ",(N(1)*10+N(0))
330 PRINT
340 RETURN
350 REM SET TIME
360 XBY(P4)=80H : REM SET PORTS A,B,&C AS OUTPUT
370 REM MSB OF REG 5 12(0)/24(1) HRS & MSB-1 AM(0)/PM(1)
380 FOR A=0 TO 12
390 PRINT "REGISTER",A, : INPUT X
400 XBY(P2)=A : XBY(P1)=X
405 REM WRITE STROBE
410 XBY(P3)=10H : XBY(P3)=50H : XBY(P3)=10H : XBY(P3)=00H
420 NEXT A
430 XBY(P4)=90H : REM RESTORE READ PORT SETTINGS
440 PRINT
450 RETURN

```

(continued)

listing2.386

TEXT

Circuit Cellar: "Real-Time Clocks: A View Toward the Future." See listing1.386 for details.

```

E
10  REM  APPLICATION PROGRAM USING ONLY BASIC TO DEMONSTRATE
20  REM  SMARTWATCH REAL TIME CLOCK ON BCC52 COMPUTER CONTROLLER BOARD
30  CLEAR
40  STRING 200,15
50  $(1)="SUNDAY"
60  $(2)="MONDAY"
70  $(3)="TUESDAY"
80  $(4)="WEDNESDAY"
90  $(5)="THURSDAY"
100 $(6)="FRIDAY"
110 $(7)="SATURDAY"
120  REM
130  REM ***** MAIN MENU *****
140  REM
150  PRINT "0=READ DATE/TIME  1=ENTER NEW DATE/TIME ?"
160  G=GET
170  GOSUB 1350 : REM  GET NUMBER 0-9
180  PRINT CHR(18),CHR(27),"Y" : REM  CLR & HOME TERMITE TERMINAL
190  IF G=0 THEN GOSUB 790 : REM  READ & DISPLAY DATE/TIME INFO
200  IF G=1 THEN GOSUB 250 : REM  GATHER & SAVE NEW DATE/TIME INFO
210  GOTO 150
220  REM
230  REM ***** GATHER $ SAVE NEW DATE/TIME INFO *****
240  REM
250  J=XBY(4000H) : REM  SAVE BYTE LOCATED IN 4000H TO REPLACE WHEN DONE
260  GOSUB 1420 : REM  SEND PATTERN RECOGNITION CODES
270  PRINT "ENTER DATE  MMDDYY"
280  G=GET
290  FOR Z=6 TO 8 : REM  USE G(6) FOR MM, G(7) FOR DD, G(8) FOR YY
300  GOSUB 1350 : REM  GET NUMBER 0-9
310  PRINT G, : REM  ECHO NUMBER 0-9
320  H=G*16 : REM  STORE NUMBER IN UPPER NIBBLE
330  GOSUB 1350
340  PRINT G,
350  G(Z)=H+G : REM  COMBINE NUMBERS 1 IN UPPER NIBBLE, 1 IN LOWER NIBBLE
360  NEXT Z
370  PRINT
380  G=G(6) : REM
390  G(6)=G(7) : REM  SWAP 6 & 7, NOW 6,7,8 IN DD/MM/YY
400  G(7)=G : REM
410  G(1)=0 : REM  SET TENTHS & HUNDREDS OF A SECOND = 0
420  PRINT "DAY OF THE WEEK SUN=0 MON=1 TUE=2 WED=3 THU=4 FRI=5 SAT=6 ?"
430  G=GET
440  GOSUB 1350
450  PRINT G
460  PRINT
470  G(5)=G.OR.10H : REM  OR BIT4 TO IGNORE RESET FROM PIN 1
480  PRINT "ENTER TIME  HHMMSS"
490  G=GET
500  FOR Z=4 TO 2 STEP -1 : REM  USE G(4) FOR HH, G(3) FOR MM, G(2) FOR SS
510  GOSUB 1350
520  PRINT G,
530  H=G*16
540  GOSUB 1350
550  PRINT G,
560  G(Z)=H+G
570  NEXT Z
580  PRINT
590  PRINT "IS THE TIME IN  0=24 HOUR FORMAT  1=12 HOUR FORMAT  ?"
600  G=GET
610  GOSUB 1350
620  IF G<>1 THEN 680 : REM  IF NOT 1 THEN JUMP
630  G(4)=(G(4).OR.80H) : REM  OR BIT7 TO INDICATE 12 HOUR FORMAT
640  PRINT "IS IT  0=AM  1=PM  ?"
650  G=GET
660  GOSUB 1350
670  IF G=1 THEN G(4)=(G(4).OR.20H) : REM  OR BITS TO INDICATE PM

```



```

680 REM HOLD FOR TIME SYNCHRONIZATION
690 PRINT "HIT '0' TO GO SET THE NEW DATE/TIME"
700 GOSUB 1350
710 IF G<>0 THEN 700
720 GOSUB 1530 : REM STORE DATE/TIME INFO TO SMARTWATCH
730 XBY(4000H)=J : REM REPLACE BYTE TO 4000H
740 G=0
750 RETURN
760 REM
770 REM ***** READ & DISPLAY DATE/TIME *****
780 REM
790 J=XBY(4000H)
800 GOSUB 1420 : REM SEND PATTERN RECOGNITION CODES
810 GOSUB 1230 : REM READ SMARTWATCH REGISTERS
820 PRINT "TODAY IS ",$(G(5).AND.7H)+1) : REM STRIP OFF DAY OF WEEK
830 $(8)=" / / " : REM INITIALIZE DATE STRING
840 Z=7 : REM USE G(7) MM REGISTER
850 X=1 : REM PLUG CHARACTERS INTO STRING STARTING AT POSITION 1
860 GOSUB 1630 : REM GET 2 CHARACTERS FROM G(Z) AND PLUG INTO STRING $(8)
870 Z=6
880 X=4
890 GOSUB 1630
900 Z=8
910 X=7
920 GOSUB 1630
930 $(9)=$(8) : REM SAVE IT IN $(9) FOR ANY FUTURE USE
940 PRINT $(9)
950 $(8)=" " : REM INITIALIZE TIME STRING
960 G(9)=G(4)
970 IF (G(4).AND.80H)=0 THEN 1020 : REM IF BIT7=0 THEN 24 HR FORMAT, JUMP
980 IF (G(4).AND.20H)=0 THEN ASC$(8),13)=41H : REM IF BIT5 = 0, PLUG A
990 IF (G(4).AND.20H)=20H THEN ASC$(8),13)=50H : REM IF BIT5 SET, PLUG P
1000 ASC$(8),14)=4DH : REM PLUG M
1010 G(9)=(G(4).AND.1FH) : REM STRIP OFF FORMAT FROM HOUR REGISTER
1020 Z=9
1030 X=1
1040 GOSUB 1630
1050 ASC$(8),3)=3AH : REM PLUG IN THE CHARACTER FOR COLON
1060 Z=3
1070 X=4
1080 GOSUB 1630
1090 ASC$(8),6)=3AH
1100 Z=2
1110 X=7
1120 GOSUB 1630
1130 Z=1
1140 X=10
1150 GOSUB 1630
1160 PRINT $(8)
1170 XBY(4000H)=J
1180 G=0
1190 RETURN
1200 REM
1210 REM ***** READ SMARTWATCH REGISTERS *****
1220 REM
1230 FOR Z=1 TO 8
1240 G(Z)=0
1250 FOR X=1 TO 8
1260 G=(XBY(4000H).AND.1) : REM G = BIT0
1270 IF G=0 THEN 1290 : REM BIT = 0, DON'T ADD ANYTHING TO REGISTER BYTE
1280 G(Z)=G(Z)+(2**(X-1)) : REM BUILD REGISTER BYTE FROM BITS RECEIVED
1290 NEXT X
1300 NEXT Z
1310 RETURN
1320 REM
1330 REM ***** GET NUMBER 0-9 *****
1340 REM
1350 G=GET
1360 IF G<48.OR.G>57 THEN 1350
1370 G=G-48 : REM ASC TO 0-9
1380 RETURN
1390 REM
1400 REM ***** INITIALIZE PATTERN RECOGNITION CODES *****
1410 REM
1420 G(1)=0C5H
1430 G(2)=3AH

```

(continued)

```

1440 G(3)=0A3H
1450 G(4)=5CH
1460 G(5)=0C5H
1470 G(6)=3AH
1480 G(7)=0A3H
1490 G(8)=5CH
1500 REM
1510 REM ***** SEND REGISTERS TO SMARTWATCH *****
1520 REM
1530 FOR Z=1 TO 8
1540 FOR X=1 TO 8
1550 IF (G(Z).AND.(2**(X-1)))<>0 THEN G=1 ELSE G=0 : REM STRIP OFF BIT
1560 XBY(4000H)=G : REM SEND BIT TO SMARTWATCH
1570 NEXT X
1580 NEXT Z
1590 RETURN
1600 REM
1610 REM ***** GET 2 CHARACTERS FROM G(Z) REGISTER, PLUG $(8) @ X
1620 REM
1630 G=INT(G(Z)/16)
1640 ASC$(8,X)=G+48
1650 ASC$(8,X+1)=G(Z)-(G*16)+48
1660 RETURN
1670 REM
1680 REM ***** END *****

```

explorer.txt

TEXT

Programming Insight, "Macintosh Explorer," Olav Andrade.
March, page 145.

```

0EM This program is to be made available at no charge.
REM Macintosh Explorer
REM Author: O. Andrade CServe: 74726,1177
REM 14 Shanley St.
REM Kitchener, Ontario
REM Canada
REM N2H 5N8

```

```

DEFINT A-Z:GOSUB 550
CLS:ON ERROR GOTO 4200:GOTO 2560
230 C2#=A0#:GOSUB 370:W=N
FOR I6=0 TO 3
C2#=C2#+2
GOSUB 370:W0(I6)=N
U0(I6)=F
NEXT I6
N6=F:N3=F
N2=W0(0):N4=W0(1):GOSUB 420
K=0:WHILE K<4 AND U0(K):K=K+1:WEND
A0#=A0#+2+2*K
RETURN
370 N=PEEK(C2#):IF N>127 THEN N=N OR &HFF00
N=N*256+PEEK(C2#+1):RETURN

420 I4=0:WHILE((I4<06)AND(W AND 02(I4))<>03(I4)):I4=I4+1:WEND
IF I4>=06 GOTO 500
S4=0:04$="":00$=01$(I4):B=F
IF 05(I4)>0 THEN ON 05(I4)GOSUB
1460,1470,1560,1480,1490,1520,1730,1740,1750
IF 05(I4)>10 THEN ON 05(I4)-10 GOSUB
2230,1770,1800,1570,1840,1870,1880,1900,1930,1970
IF 05(I4)>20 THEN ON 05(I4)-20 GOSUB
1990,2300,2020,2030,2050,2080,1600,2120,2320,1680
IF 05(I4)>30 THEN ON 05(I4)-30 GOSUB 1940,2360,2370,2380,2390,2400
IF NOT B THEN PRINT A0#;" ";00$;" ";04$:GOTO 510
500 PRINT A0#;" dw $";HEX$(W):FOR I6=0 TO 3:U0(I6)=F:NEXT
510 RETURN

550 06=71:DIM 02(06),03(06),01$(06),05(06)
FOR I=0 TO 06-1:READ 01$(I),02(I),03(I),05(I):NEXT
DATA reset, -1, 20080, 1, nop, -1, 20081, 1, stop, -1,

```



```

20082, 1, rte, -1, 20083, 1
DATA      rts, -1, 20085, 1, trapv, -1, 20086, 1, rtr, -1, 20087, 1,
swap, -8, 18496, 2
DATA      "ext.w", -8, 18560, 2, "ext.l", -8, 18624, 2, link, -8, 20048,
3, unlk, -8, 20056, 4
DATA      move, -8, 20064, 35, move, -8, 20072, 36, trap, -16, 20032, 5
DATA      nbcd, -64, 18432, 6, pea, -64, 18496, 6, move, -64, 16576, 32
DATA      move, -64, 17600, 34, move, -64, 18112, 33, tas, -64, 19136, 6
DATA      jsr, -64, 20096, 6, jmp, -64, 20160, 6, exg, -3592, -16064, 7
DATA      exg, -3592, -16056, 8, exg, -3592, -15992, 9, sbcd, -3600, -
32512, 10
DATA      abcd, -3600, -16128, 10, movem, -128, 18560, 11, movem, -128,
19584, 11
DATA      db, -3848, 20680, 12, statBit, -256, 2048, 13, tst, -256,
18944, 14
DATA      clr, -256, 16896, 14, neg, -256, 17408, 14, or, -256, 0, 30
DATA      not, -256, 17920, 14, and, -256, 512, 30, bsr, -256, 24832, 15
DATA      sub, -256, 1024, 30, cmpm, -3784, -20216, 16, add, -256, 1536,
30
DATA      eor, -256, 2560, 30, cmp, -256, 3072, 30, negx, -256, 16384,
14
DATA      subx, -3792, -28416, 17, movep, -4040, 8, 18, mulu, -3684, -
16192, 19
DATA      muls, -3648, -15936, 19, chk, -3648, 16768, 19, divu, -3648, -
32576, 19
DATA      divs, -3648, -32320, 19, lea, -3648, 16832, 31, addx, -3792, -
12032, 17
DATA      memShifts, -1856, -8000, 20, s, -3904, 20672, 21, dynBit, -
3840, 256, 22
DATA      eor, -3840, -20224, 23, addq, -3840, 20480, 24, subq, -3840,
20736, 24
DATA      moveq, -3840, 28672, 25, b, -4096, 24576, 26, "move.b", -4096,
4096, 27
DATA      sub, -4096, -28672, 28, "move.l", -4096, 8192, 27, cmp, -4096,
-20480, 28
DATA      or, -4096, -32768, 28, add, -4096, -12288, 28, "move.w", -
4096, 12288, 27
DATA      dataRegShifts, -4096, -8192, 29, and, -4096, -16384, 28
A5=ASC("0"):A6=ASC("a"):F=0:T=NOT F
DIM S3$(3):S3$(0)="b":S3$(1)="w":S3$(2)="l"
DIM W0(3),U0(3),C0$(16),B3$(4),S2$(4),M(16)
C0$(0)="t":C0$(1)="f":C0$(2)="hi":C0$(3)="ls"
C0$(4)="cc":C0$(5)="cs":C0$(6)="ne":C0$(7)="eq"
C0$(8)="vc":C0$(9)="vs":C0$(10)="pl":C0$(11)="mi"
C0$(12)="ge":C0$(13)="lt":C0$(14)="gt":C0$(15)="le"
B3$(0)="btst":B3$(1)="bchg":B3$(2)="bclr":B3$(3)="bset"
S2$(0)="as":S2$(1)="ls":S2$(2)="rox":S2$(3)="ro"
CALL TEXTFONT(4):CALL TEXTSIZE(9)
B5=B5:B4=15:S=278
DIM I8(4):I.=5:I00=5:I8(0)=I00:I8(1)=I.:I8(2)=I8(0)+B4:I8(3)=I8(1)+B5
DIM M0(4):M2=I.+B5+20:M3=I00:M0(0)=M3:M0(1)=M2:M0(2)=M0(0)+B4:M0(3)=M0(1)+B5
DIM
D7(4):D.=M2+B5+20:D00=I00:D7(0)=D00:D7(1)=D.:D7(2)=D7(0)+B4:D7(3)=D7(1)+B5
DIM D(4):D1=I.:D2=I00+B4+5:D(0)=D2:D(1)=D1:D(2)=D(0)+B4:D(3)=D(1)+B5
DIM
D02(4):D04=M2:D05=D2:D02(0)=D05:D02(1)=D04:D02(2)=D02(0)+B4:D02(3)=D02(1)+B5
DIM L0(4):L2=D.:L3=D2:L0(0)=L3:L0(1)=L2:L0(2)=L0(0)+B4:L0(3)=L0(1)+B5
DIM A1(4):A3=D.+B5+20:A4=D2:A1(0)=A4:A1(1)=A3:A1(2)=A1(0)+B4:A1(3)=A1(1)+B5
DIM V(4):V2=A3+B5+10:V3=D2:V(0)=V3:V(1)=V2:V(2)=V(0)+B4:V(3)=V(1)+B5
DIM C1(4):C1(0)=0:C1(1)=0:C1(2)=V(2)+4:C1(3)=V(3)
DIM I01(1),I7(1):N1=1
DIM D9(310):D5=0:D3=0:B2=52:FOR I5=0 TO 27:READ D9(I5):NEXT
DATA &h4e56, 0, 16890, 50, 12668, -5, 24
DATA 12668, 1, 22, 12668, 1, 44, 8572, 0, 0, 46, 8572, 0
DATA 512, 36, 17402, 60, 8521, 32, &ha002, &h4e5e, &h4e75
U=0:I9=1:M1=2:D8=3:D0=4
D03=5:A2=6:V0=7:L1=8
S1=M1:RETURN
1100 E$=""
ON M4+1 GOSUB 1130,1140,1150,1160,1170,1180,1190,1250
RETURN
1130 E$="d"+CHR$(R0+A5):RETURN
1140 E$="a"+CHR$(R0+A5):RETURN
1150 E$="(a"+CHR$(R0+A5)+)":RETURN
1160 E$="(a"+CHR$(R0+A5)+)":RETURN
1170 E$="-(a"+CHR$(R0+A5)+)":RETURN

```

(continued)


```

1180 E$=STR$(N2)+" (a"+CHR$(R0+A5)+")":N6=T:RETURN
1190 IF N2<0 THEN E$="a"ELSE E$="d"
E$=E$+CHR$((N2 AND &H7000)\4096+A5)
IF N2 AND &H800 THEN E$=E$+".1"
E$="(a"+CHR$(R0+A5)+", "+E$+)"
IF(N2 AND &H80)THEN E$=STR$((N2 AND &HFF)OR &HFF00)+E$ELSE E$=STR$(N2 AND
&HFF)+E$
N6=T:RETURN
1250 ON R0+1 GOSUB 1280,1290,1310,1320,1380,1270,1270,1270
RETURN
1270 B=T:RETURN
1280 E$="("$"+HEX$(N2)+")":N6=T:RETURN
1290 GOSUB 1410:E$="("$"+HEX$(N2)+I3$+)"
N6=T:N3=T:RETURN
1310 E$=STR$(N2)+"(pc)":N6=T:RETURN
1320 IF N2<0 THEN E$="a"ELSE E$="d"
E$=E$+CHR$((N2 AND &H7000)\4096+A5)
IF N2 AND &H800 THEN E$=E$+".1"
E$="(pc, "+E$+)"
IF(N2 AND &H80)THEN E$=STR$((N2 AND &HFF)OR &HFF00)+E$ELSE E$=STR$(N2 AND
&HFF)+E$
N6=T:RETURN
1380 E$="#$"+HEX$(N2):N6=T
IF S4=2 THEN GOSUB 1410:E$=E$+I3$:N3=T
RETURN
1410 I3$=HEX$(N4):IF LEN(I3$)<4 THEN I3$=STRING$(4-LEN(I3$),"0")+I3$
1420 RETURN
1460 O4$="":RETURN
1470 O4$="d"+CHR$((W AND 7)+A5):RETURN
1480 O4$="a"+CHR$((W AND 7)+A5):RETURN
1490 IF((W AND 15)>9)THEN 1510
O4$=CHR$((W AND 15)+A5):RETURN
1510 O4$=CHR$((W AND 15)+A6-10):RETURN
1520 M4=(W AND &H38)\8:R0=(W AND 7)
GOSUB 1100:O4$=E$:IF N6 THEN U0(0)=T
IF N3 THEN U0(1)=T
RETURN
1560 O4$="a"+CHR$((W AND 7)+A5)+", "+STR$(N2):U0(0)=T:RETURN
1570 S4=(W AND &HC0)\64:M4=(W AND &H38)\8:R0=(W AND 7):GOSUB
1100:O4$=E$:O0$=O0$+S3$(S4)
1580 IF N6 THEN U0(0)=T:IF N3 THEN U0(1)=T
1590 RETURN
1600 M4=(W AND &H38)\8:R0=(W AND 7)
1610 IF(W AND &HF000)=&H1000 THEN S4=0ELSE IF(W AND &HF000)=&H2000 THEN
S4=2ELSE S4=1
1620 GOSUB 1100:S0$=E$
1630 IF N6 THEN U0(0)=T:N6=F:IF N3 THEN U0(1)=T:N3=F:N2=W0(2):N4=W0(3)ELSE
N2=N4:N4=W0(2)
1640 M4=(W AND &H1C0)\64:R0=(W AND &HE00)\512:GOSUB 1100:O4$=S0$+", "+E$
1650 IF N6 THEN IF U0(0)THEN IF U0(1)THEN U0(2)=TELSE U0(1)=TELSE U0(0)=T
1660 IF N3 THEN IF U0(1)THEN IF U0(2)THEN U0(3)=TELSE U0(2)=TELSE U0(1)=T
1670 RETURN
1680 S4=(W AND &HC0)\64:M4=(W AND &H38)\8:R0=(W AND
7):O4$="#$"+HEX$(N2):U0(0)=T:IF S4<>2 THEN O4$=O4$+", "ELSE GOSUB
1410:N3=T:O4$=O4$+I3$+", "
1690 IF((W AND &H3F)=&H3C)THEN O4$=O4$+"sr":GOTO 1700ELSE IF NOT N3 THEN
N2=W0(1):N4=W0(2)ELSE U0(1)=T:N3=F:N2=W0(2):N4=W0(3)
1695 GOSUB 1100:O4$=O4$+E$
1700 O0$=O0$+S3$(S4):IF N6 THEN IF NOT U0(1)THEN U0(1)=T:IF NOT N3 THEN
RETURNELSE U0(2)=TELSE U0(2)=T:IF N3 THEN U0(3)=T
1710 RETURN
1720 O4$="d"+CHR$((W AND &HE00)\512+A5)+", d"+CHR$((W AND 7)+A5):RETURN
1730 O4$="a"+CHR$((W AND &HE00)\512+A5)+", a"+CHR$((W AND 7)+A5):RETURN
1740 O4$="d"+CHR$((W AND &HE00)\512+A5)+", a"+CHR$((W AND 7)+A5):RETURN
1750 IF(W AND 8)THEN 1760ELSE O4$="d"+CHR$((W AND 7)+A5)+", d"+CHR$((W AND
&HE00)\512+A5):RETURN
1760 O4$="-(a"+CHR$((W AND 7)+A5)+", -(a"+CHR$((W AND
&HE00)\512+A5)+)":RETURN
1770 O0$=O0$+C0$((W AND &HF00)\256):O4$="d"+CHR$((W AND 7)+A5)+", $"
1780 U0(0)=T:IF N2<0 THEN O4$=O4$+STR$(N2)ELSE O4$=O4$+" "+STR$(N2)
1790 RETURN
1800 O0$=B3$((W AND &H60)\64):M4=((W AND &H38)\8):R0=(W AND 7)
1810 U0(0)=T:O4$="#$"+STR$(&H1F AND N2))+", ":N2=N4:N4=W0(2):GOSUB
1100:O4$=O4$+E$
1820 IF N6 THEN U0(1)=T:IF N3 THEN U0(2)=T
1830 RETURN
1840 IF(W AND 128)THEN O0$=O0$+".s": O4$="$ "+STR$((W AND &HFF)OR &HFF00)+":

```



```

"+STR$(A0#*2-(W AND &HFF)):RETURN
1850 IF(W AND &HFF)=0 THEN U0(0)=T:IF N2<0 THEN O4$="$ "+STR$(N2)+";
"+STR$(A0#*4+N2):RETURN ELSE O4$="$ "+STR$(N2)+"; "+STR$(A0#*4+N2):RETURN
1860 O0$=O0$+".s": O4$="$ "+STR$(W AND &HFF)+"; "+STR$(A0#*2+(W AND
&HFF)):RETURN
1870 O0$=O0$+S3$((W AND &HC0)\128):O4$="(a"+CHR$((W AND 7)+A5)+")+";
(a"+CHR$((W AND &HE00)\512+A5)+")+":RETURN
1880 IF(W AND 8)THEN 1890ELSE O4$="d"+CHR$((W AND 7)+A5)+"; d"+CHR$((W AND
&HE00)\512+A5):RETURN
1890 O4$="(a"+CHR$((W AND 7)+A5)+"), -(a"+CHR$((W AND
&HE00)\512+A5)+")":RETURN
1900 U0(0)=T:IF(W AND 64)THEN O0$=O0$+".I"ELSE O0$=O0$+".w"
1910 IF(W AND 128)THEN O4$="d"+CHR$((W AND
&HE00)\512+A5)+"; "+STR$(N2)+"(a"+CHR$((W AND 7)+A5)+")":RETURN
1920 O4$=STR$(N2)+"(a"+CHR$((W AND 7)+A5)+"; d"+CHR$((W AND
&HE00)\512+A5):RETURN
1930 M4=(W AND &H38)\8:R0=(W AND 7):GOSUB 1100:O4$=E$+", d"+CHR$((W AND
&HE00)\512+A5):GOTO 1950
1940 M4=(W AND &H38)\8:R0=(W AND 7):GOSUB 1100:O4$=E$+", a"+CHR$((W AND
&HE00)\512+A5)
1950 IF N6 THEN U0(0)=T:IF N3 THEN U0(1)=T
1960 RETURN
1970 O0$=S2$((W AND &H600)\512):IF(W AND 256)THEN O0$=O0$+".I"ELSE O0$=O0$+".r"
1980 M4=(W AND &H38)\8:R0=(W AND 7):GOSUB 1100:O4$=E$:GOSUB 1580:RETURN
1990 O0$=O0$+C0$((W AND &HF00)\256):M4=(W AND &H38)\8:R0=(W AND 7):GOSUB
1100:O4$=E$:GOSUB 1580:RETURN
2000 O0$=B3$((W AND &H60)\64):O4$="d"+CHR$((W AND &HE00)\512+A5)+"; "
2010 M4=(W AND &H38)\8:R0=(W AND 7):GOSUB 1100:O4$=O4$+E$:GOSUB 1580:RETURN
2020 O4$="d"+CHR$((W AND &HE00)\512+A5):GOTO 2040
2030 t0=(W AND &HE00)\512: IF t0=0 THEN O4$="#8" ELSE O4$="#"+CHR$(t0+A5)
2040 S4=(W AND &HC0)\64 :O0$=O0$+S3$(S4):M4=(W AND &H38)\8:R0=(W AND 7):GOSUB
1100:O4$=O4$+", "+E$:GOSUB 1580:RETURN
2050 O4$="d"+CHR$((W AND &HE00)\512+A5)
2060 IF(W AND 128)THEN O4$="#"+STR$((W AND &HFF)OR &HFF00)+O4$:RETURN
2070 O4$="#"+STR$(W AND &HFF)+O4$:RETURN
2080 O0$=O0$+C0$((W AND &HF00)\256):IF(W AND 128)THEN O0$=O0$+".s":
O4$="$ "+STR$((W AND &HFF)OR &HFF00)+"; "+STR$(A0#*2-(W AND &HFF)):RETURN
2090 IF(W AND &HFF)=0 THEN U0(0)=T:O0$=O0$+".w": IF N2<0 THEN O4$="$
"+STR$(N2)+"; "+STR$(A0#*4+N2):RETURN ELSE O4$="$ "+STR$(N2)+";
"+STR$(A0#*4+N2):RETURN
2100 O0$=O0$+".s": O4$="$ "+STR$(W AND &HFF)+"; "+STR$(A0#*2+(W AND
&HFF)):RETURN
2110 RETURN
2120 O=(W AND &HE0)\64+1:ON O GOSUB 2160,2170,2180,2160,2170,2180,2170,2180
2130 ON O GOSUB 2190,2190,2190,2200,2200,2200,2210,2210
2140 IF N6 THEN U0(0)=T:IF N3 THEN U0(1)=T
2150 RETURN
2160 O0$=O0$+".b":S4=0:RETURN
2170 O0$=O0$+".w":S4=1:RETURN
2180 O0$=O0$+".I":S4=2:RETURN
2190 GOSUB 2220:O4$=E$+", d"+CHR$((W AND &HE00)\512+A5):RETURN
2200 GOSUB 2220:O4$="d"+CHR$((W AND &HE00)\512+A5)+"; "+E$:RETURN
2210 GOSUB 2220:O4$=E$+", a"+CHR$((W AND &HE00)\512+A5):RETURN
2220 M4=(W AND &H38)\8:R0=(W AND 7):GOSUB 1100:RETURN
2230 IF(W AND 64)THEN O0$=O0$+".I":S4=2ELSE O0$=O0$+".w":S4=1
2240 R2=N2:U0(0)=T:N2=N4=W0(2)
2250 M4=(W AND &H38)\8:R0=(W AND 7):GOSUB 1100
2260 IF N6 THEN U0(1)=T:IF N3 THEN U0(2)=T
2270 IF M4=4 THEN FOR I1=1 TO 15:M(I1)=2^(15-I1):NEXT:M(0)=-32768!ELSE FOR
I1=0 TO 14:M(I1)=2^I1:NEXT:M(15)=-32768!
2280 GOSUB 2440:IF(W AND &HF00)=&H800 THEN O4$=O4$+", "+E$ELSE O4$=E$+", "+O4$
2290 RETURN
2300 O4$="d"+CHR$((W AND &HE00)\512+A5)
2310 O0$=B3$((W AND &HC0)\64):M4=(W AND &H38)\8:R0=(W AND 7):GOSUB
1100:O4$=O4$+", "+E$:GOSUB 1580:RETURN
2320 IF(W AND 32)THEN O4$="d"+CHR$((W AND &HE00)\512+A5)ELSE t0=(W AND
&HE00)\512: IF t0=0 THEN O4$="#8" ELSE O4$="#"+CHR$(t0+A5)
2330 O4$=O4$+", d"+CHR$((W AND 7)+A5)
2340 O0$=S2$((W AND &H18)\8):IF(W AND 256)THEN O0$=O0$+".I"ELSE O0$=O0$+".r"
2350 O0$=O0$+S4$((W AND &HC0)\64):RETURN
2360 GOSUB 1520:O4$="sr", "+O4$:RETURN
2370 GOSUB 1520:O4$=O4$+", sr":RETURN
2380 GOSUB 1520:O4$=O4$+", ccr":RETURN
2390 GOSUB 1480:O4$=O4$+", usp":RETURN
2400 GOSUB 1480:O4$="usp", "+O4$:RETURN
2410
2420 REM return movem operand (extension word decoding)

```

(continued)


```

2430
2440 FOR I1=0 TO 1
2450 F0=T:IF I1=0 THEN O4$="":R$="d"ELSE R$="a":IF O4$<>" "THEN IF M(15)<0
AND(R2 AND &HFF00)<>0 THEN O4$=O4$+ "/"ELSE IF M(15)>0 AND(R2 AND &HFF)<>0 THEN
O4$=O4$+ "/"
2460 I2=0:WHILE I2<8
2470 IF (M(I2+I1*8)AND R2)=0 THEN J0=I2+1:GOTO 2510
2480 IF F0 THEN O4$=O4$+R$+CHR$(I2+A5):F0=FALSE O4$=O4$+ ", "+R$+CHR$(I2+A5)
2490 J0=I2+1:WHILE J0<8 AND(M(J0+I1*8)AND R2):J0=J0+1:WEND
2500 IF (M(J0+I1*8-1)AND R2)AND J0>I2+1 THEN O4$=O4$+ "-" +R$+CHR$(J0+A5-1)
2510 I2=J0:WEND
2520 NEXT:RETURN
2530
2540 REM main loop
2550
2560 L=F:GOSUB 2950
2570 WHILE NOT L
2580 GOSUB 2650
2590 ON R1 GOSUB 2820,2820,2820,3130,3230,3420,3450,2615
2600 WEND
2610 END
2615 IF MOUSE(0)<>0 THEN L=T:SYSTEMELSE RETURN
2620
2630 REM determine mouse region
2640
2650 D06=MOUSE(1)-I.:D07=MOUSE(2)-I00:IF 0<=D06 AND D06<B5 AND 0<=D07 AND
D07<=B4 THEN R1=I9:D01=MOUSE(0):RETURN
2660 D06=MOUSE(1)-M2:D07=MOUSE(2)-M3:IF 0<=D06 AND D06<B5 AND 0<=D07 AND
D07<=B4 THEN R1=M1:D01=MOUSE(0):RETURN
2670 D06=MOUSE(1)-D.:D07=MOUSE(2)-D00:IF 0<=D06 AND D06<B5 AND 0<=D07 AND
D07<=B4 THEN R1=D8:D01=MOUSE(0):RETURN
2680 D06=MOUSE(1)-D1:D07=MOUSE(2)-D2:IF 0<=D06 AND D06<B5 AND 0<=D07 AND
D07<=B4 THEN R1=D0:D01=MOUSE(0):RETURN
2690 D06=MOUSE(1)-D04:D07=MOUSE(2)-D05:IF 0<=D06 AND D06<B5 AND 0<=D07 AND
D07<=B4 THEN R1=D03:D01=MOUSE(0):RETURN
2695 D06=MOUSE(1)-L2:D07=MOUSE(2)-L3:IF 0<=D06 AND D06<B5 AND 0<=D07 AND
D07<=B4 THEN R1=L1:D01=MOUSE(0):RETURN
2700 D06=MOUSE(1)-A3:D07=MOUSE(2)-A4:IF 0<=D06 AND D06<B5 AND 0<=D07 AND
D07<=B4 THEN R1=A2:D01=MOUSE(0):RETURN
2710 D06=MOUSE(1)-V2:D07=MOUSE(2)-V3:IF 0<=D06 AND D06<B5 AND 0<=D07 AND
D07<=B4 THEN R1=V0:D01=MOUSE(0):RETURN
2720 R1=U:D01=MOUSE(0):RETURN
2730
2740 REM select a path
2750
2760 ON S1 GOSUB 2780,2790,2800
2770 RETURN
2780 CALL INVERTRECT(VARPTR(I8(0))):RETURN
2790 CALL INVERTRECT(VARPTR(M0(0))):RETURN
2800 CALL INVERTRECT(VARPTR(D7(0))):RETURN
2810
2820 IF MOUSE(0)=0 THEN RETURNELSE ON S1 GOSUB 2880,2890,2900
2830 ON R1 GOSUB 2850,2860,2870
2840 S1=R1:WHILE MOUSE(0)=1:WEND:GOSUB 2940:RETURN
2850 A0#=C:RETURN
2860 A0#=M5#:RETURN
2870 A0#=D4#:RETURN
2880 C=A0#:RETURN
2890 M5#=A0#:RETURN
2900 D4#=A0#:RETURN
2910
2920 REM draw screen controls
2930
2940 CALL ERASERECT(VARPTR(C1(0)))
2950 CALL MOVETO(I8(1)+10,I8(2)-5):PRINT"input";:CALL FRAMERECT(VARPTR(I8(0)))
2960 CALL MOVETO(M0(1)+10,M0(2)-5):PRINT"memory";:CALL
FRAMERECT(VARPTR(M0(0)))
2970 CALL MOVETO(D7(1)+10,D7(2)-5):PRINT"disk";:CALL FRAMERECT(VARPTR(D7(0)))
2980 CALL MOVETO(D(1)+10,D(2)-5):PRINT"disassemble";:CALL
FRAMERECT(VARPTR(D(0)))
2990 CALL MOVETO(D02(1)+10,D02(2)-5):PRINT"dump";:CALL
FRAMERECT(VARPTR(D02(0)))
2995 CALL MOVETO(L0(1)+10,L0(2)-5):PRINT"quit";:CALL FRAMERECT(VARPTR(L0(0)))
3000 CALL MOVETO(A1(1)+5,D7(2)-5):PRINT"address";
3010 CALL MOVETO(V(1)+5,D7(2)-5):PRINT"value";
3020 CALL MOVETO(A1(1)+5,A1(2)-5):PRINT A0#;:CALL FRAMERECT(VARPTR(A1(0)))
3030 CALL MOVETO(V(1)+5,V(2)-5):ON S1 GOSUB 3050,3070,3080:PRINT V1$;:CALL

```



```

FRAMERECT(VARPTR(V(0)))
3040 GOSUB 2760:RETURN
3050 IF A0#>=N1 THEN V1$="?"ELSE IF P THEN V1$=STR$(I01(A0#))ELSE
V1$=STR$(I7(A0#))
3060 RETURN
3070 C2#=A0#:GOSUB 370:V1$=STR$(N):RETURN
3080 IF 0>A0#-D5*256 OR A0#-D5*256>=2*D3 THEN V1$="?"ELSE V1$=STR$(D9((A0#-
D5*256)\2+B2))
3090 RETURN
3100
3110 REM mouse in disassemble button
3120
3130 IF MOUSE(0)=0 THEN RETURN
3140 WHILE MOUSE(0)<>0
3150 CALL MOVETO(0,S)
3160 ON S1 GOSUB 3730,230,3990: 'pick a path
3170 GOSUB 2940:D01=MOUSE(0)
3180 WEND
3190 RETURN
3200
3210 REM mouse in dump button
3220
3230 IF MOUSE(0)=0 THEN RETURN
3240 WHILE MOUSE(0)<>0
3250 CALL MOVETO(0,S)
3260 ON S1 GOSUB 3820,3330,4100
3270 GOSUB 2940:D01=MOUSE(0)
3280 WEND
3290 RETURN
3300
3310 REM dump input path
3320
3330 I2$="":PRINT A0#;" ";
3340 FOR I0=0 TO 15
3350 I1=PEEK(A0#):IF I1<16 THEN PRINT"0";HEX$(I1);ELSE PRINT HEX$(I1);
3360 IF ASC(" ")<I1 AND I1<128 THEN I2$=I2$+CHR$(I1)ELSE I2$=I2$+"."
3370 A0#=A0#+1:NEXT
3380 PRINT" ";I2$:RETURN
3390
3400 REM mouse in address box
3410
3420 IF MOUSE(0)=0 THEN RETURN
3430 CALL INVERTRECT(VARPTR(A1(0))):CALL MOVETO(A1(1)+5,A1(2)-5):B1=T:GOSUB
3520
3440 CALL INVERTRECT(VARPTR(A1(0))):D01=MOUSE(0):A0#=N5#:GOSUB 2940:RETURN
3450 IF MOUSE(0)=0 THEN RETURN
3460 CALL INVERTRECT(VARPTR(V(0))):CALL MOVETO(V(1)+5,V(2)-5):ON S1 GOSUB
3650,3645,3645
3470 D01=MOUSE(0):IF N5#>32767 THEN N5#=N5#-65536!
3480 V1=N5#:RETURN
REM fetch keys: address or input
3520 I1$=INKEY$:IF I1$=""THEN 3520
3530 IF ASC(I1$)=13 THEN IF N0 THEN N5#=-1*N5#:RETURNELSE RETURN
3540 IF B1 THEN N5#=0:B1=F:H=F:N0=F:IF I1$="$"THEN H=T:PRINT I1$;:GOTO
3520ELSE IF I1$="-"THEN PRINT I1$;:N0=T:GOTO 3520
3550 IF"0"<=I1$AND I1$<="9"THEN PRINT I1$;:A7=ASC(I1$)-ASC("0"):IF H THEN
N5#=N5#+16+A7:GOTO 3520ELSE N5#=N5#+10+A7:GOTO 3520
3560 IF"A"<=I1$AND I1$<="F"THEN IF NOT H THEN BEEP:GOTO 3520ELSE PRINT
I1$;:N5#=N5#+16+ASC(I1$)-ASC("A")+10:GOTO 3520
3570 IF"a"<=I1$AND I1$<="f"AND H THEN PRINT I1$;:N5#=N5#+16+ASC(I1$)-
ASC("a")+10:GOTO 3520
3580 BEEP:GOTO 3520

3620 P=NOT P:IF P THEN ERASE I01:DIM I01(A0#+1)ELSE ERASE I7:DIM I7(A0#+1)
3630 FOR I1=0 TO N1-1:IF P THEN I01(I1)=I7(I1)ELSE I7(I1)=I01(I1)
3640 NEXT:N1=A0#+1
3645 RETURN
REM disassemble input
3650 IF A0#>=N1 THEN GOSUB 3620
3660 B1=T:GOSUB 3520:N5#=N5#-65536!*INT(N5#/65536!):IF N5#>32767 THEN N5#=N5#-
65536!
3670 IF P THEN I01(A0#)=N5#ELSE I7(A0#)=N5#
3680 A0#=A0#+1:GOSUB 2940:RETURN
3730 IF A0#>N1 THEN W=-1ELSE IF P THEN W=I01(A0#)ELSE W=I7(A0#)
3740 P0=A0#:FOR I6=0 TO 3:P0=P0+1:IF P0>=N1 THEN W0(I6)=-1ELSE IF P THEN
W0(I6)=I01(P0)ELSE W0(I6)=I7(P0)
3750 U0(I6)=F:NEXT:N6=F:N3=F

```



```

3760 N2=W0(0):N4=W0(1):GOSUB 420
3770 K=0:WHILE K<4 AND A0#1+K<N1 AND U0(K):K=K+1:WEND
3780 A0#=A0#1+K:RETURN
REM    dump disk
3820 PRINT A0#;"          ";I2$="":FOR I0=0 TO 7
3830 IF A0#>=N1 THEN PRINT"????";I2$=I2$+" " :GOTO 3880
3840 IF P THEN I1=I01(A0#)ELSE I1=I7(A0#)
3850 I3$=HEX$(I1):IF LEN(I3$)=4 THEN PRINT I3$;ELSE PRINT STRING$(4-
LEN(I3$),"0");I3$;
3860 IF(ASC(" ")<INT(I1\256))AND(INT(I1\256)<128)THEN
I2$=I2$+CHR$(INT(I1\256))ELSE I2$=I2$+"."
3870 IF ASC(" ")<(I1 AND 255)AND(I1 AND 255)<128 THEN I2$=I2$+CHR$(I1 AND
255)ELSE I2$=I2$+"."
3880 A0#=A0#1:NEXT:PRINT"          ";I2$
3890 RETURN
REM    fetch a disk sector
3930 B0=F:D5=INT(A0#/512):IF D5 AND 64 THEN D9(15)=(D5 OR &HFF80)*512ELSE
D9(15)=(D5 AND &H7F)*512
3935 D9(14)=D5\128:D5=2*D5:D61=VARPTR(D9(0)):CALL D61
3940 D3=(D9(28+20)*256+D9(28+21))\2:IF A0#-D5*256>=2*D3 THEN B0=T
3950 RETURN
REM    disassemble disk
3990 IF A0#-D5*256<0 OR A0#-D5*256>2*D3 THEN GOSUB 3930:IF B0 THEN BEEP:RETURN
4000 P0=(A0#-256*D5)\2:A0#=2*P0+256*D5:W=D9(P0+B2)
4010 FOR I6=0 TO 3:P0=P0+1:IF P0>=D3 THEN J#=A0#:A0#=A0#+2*P0:GOSUB
3930:P0=0:A0#=J#:IF B0 THEN W0(I6)=-1:GOTO 4030
4020 W0(I6)=D9(P0+B2)
4030 U0(I6)=F:NEXT:N6=F:N3=F
4040 N2=W0(0):N4=W0(1):GOSUB 420
4050 K=0:WHILE K<4 AND U0(K):K=K+1:WEND
4060 A0#=A0#+2+2*K:RETURN
REM    dump disk
4100 IF 2*INT(A0#/2)<>A0#THEN A0#=A0#-1
4105 PRINT A0#;"          ";I2$="":FOR I0=0 TO 7
4110 IF A0#-D5*256<0 OR A0#-D5*256>=2*D3 THEN GOSUB 3930:IF B0 THEN
PRINT"????";I2$=I2$+" " :GOTO 4160
4120 I1=D9((A0#-D5*256)\2+B2)
4130 I3$=HEX$(I1):IF LEN(I3$)=4 THEN PRINT I3$;ELSE PRINT STRING$(4-
LEN(I3$),"0");I3$;
4140 IF(ASC(" ")<INT(I1\256))AND(INT(I1\256)<128)THEN
I2$=I2$+CHR$(INT(I1\256))ELSE I2$=I2$+"."
4150 IF ASC(" ")<(I1 AND 255)AND(I1 AND 255)<128 THEN I2$=I2$+CHR$(I1 AND
255)ELSE I2$=I2$+"."
4160 A0#=A0#+2:NEXT:PRINT"          ";I2$
4170 RETURN
REM    expand the input path array
4200 IF NOT(ERL=3620 AND ERR=7)THEN ON ERROR GOTO 0
4210 BEEP:BEEP=P:NOT P:IF P THEN DIM I7(1)ELSE DIM I01(1)
4220 GOSUB 2940:RESUME 2570

```

diophant.bas

TEXT

Mathematical Equations, "Diophantine Equations," Robert T. Kurosaka,
March, page 343.

```

10 '*****
20 '*      DIOPHANTINE EQUATION SOLVER      *
30 '*      BY BOB KUROSAKA                  *
40 '*****
50 CLS
60 PRINT "This program solves equations of the form AX + BY = C,"
70 PRINT "where A, B, C, X, and Y are all integer values."
80 PRINT :PRINT "Enter your equation as shown in the general form."
90 PRINT "For example, enter 154X + 69Y = 5000 or 154X - 69Y = 5000."
100 PRINT "Do not enter negative coefficients with parentheses."
110 PRINT "That is, do NOT enter 154X + (-69Y) = 5000."
120 PRINT :PRINT "The program will not work properly for the degenerate case"
130 PRINT "where either A or B is 0."
140 PRINT :INPUT "Enter equation";EQUATION$:A$=EQUATION$
150 REM DEFINE A READABLE FUNCTION FOR DISCARDING LEFTMOST CHARACTERS.
160 DEF FNDROP.LEFT$(A$)=RIGHT$(A$,LEN(A$)-1)
170 REM PARSING ROUTINE
180 A=VAL(A$)

```



```

190 IF A=0 THEN A=1:IF LEFT$(A$,1)="-" THEN A=-1
200 A$=FNDROP.LEFT$(A$)
210 DISCARD$=LEFT$(A$,1)
220 WHILE DISCARD$<>"+" AND DISCARD$<>"-"
230     A$=FNDROP.LEFT$(A$)
240     DISCARD$=LEFT$(A$,1)
250 WEND
260 B=VAL(A$)
270 IF B=0 THEN B=1:IF DISCARD$="-" THEN B=-1
280 WHILE DISCARD$<>"="
290     A$=FNDROP.LEFT$(A$)
300     DISCARD$=LEFT$(A$,1)
310 WEND
320 A$=FNDROP.LEFT$(A$)
330 C=VAL(A$)
340 IF A<>INT(A) OR B<>INT(B) OR C<>INT(C) THEN
    PRINT "NOT A DIOPHANTINE EQUATION":GOTO 760
350 REM END OF PARSING ROUTINE
360 REM EUCLIDEAN ALGORITHM FOR FINDING GCD.
370 REM FIRST, INITIALIZE THE TERMS FOR THE ALGORITHM
380 IF ABS(A)>=ABS(B) THEN DIVIDEND=A:DIVISOR=B
390 IF ABS(A)<ABS(B) THEN DIVISOR=A:DIVIDEND=B:SWAP.XY$="YES"
400 REM USE 'FIX' INSTEAD OF 'INT' TO TRUNCATE RATHER THAN ROUND NEGATIVE #s.
410 QUOTIENT=FIX(DIVIDEND/DIVISOR)
420 REMAINDER=DIVIDEND-DIVISOR*QUOTIENT
430 REM X1=ONGOING COUNT OF X', Y1=ONGOING COUNT OF Y'. YOU CAN KEEP TRACK OF
    ALL ONGOING COUNTS BY ONLY USING THE PREVIOUS 2 VALUES FOR X' AND Y', SO
    WE ONLY NEED X1, X2, X3, AND Y1, Y2, Y3.
440 X1=1:Y1=-QUOTIENT
450 REM IF EITHER A OR B IS AN EVEN MULTIPLE OF THE OTHER, THEN EITHER X' OR
    Y' WILL EQUAL 1 WHILE THE OTHER EQUALS 0.
460 IF REMAINDER=0 THEN X2=0:Y2=1:GOTO 620
470 DIVIDEND=DIVISOR:DIVISOR=REMAINDER
480 QUOTIENT=FIX(DIVIDEND/DIVISOR)
490 REMAINDER=DIVIDEND-DIVISOR*QUOTIENT
500 X2=-QUOTIENT*X1:Y2=1-QUOTIENT*Y1
510 REM IF A GCD IS FOUND ON THE SECOND ITERATION OF THE EUCLIDEAN ALGORITHM,
    THEN X'=X1, Y'=Y1. IN ALL SUBSEQUENT CASES, X'=X2, Y'=Y2.
520 IF REMAINDER=0 THEN X2=X1:Y2=Y1:GOTO 620
530 REM THE FIRST TWO ITERATIONS ARE THE ONLY ONES THAT DO NOT FOLLOW THE
    PATTERN: X(N)=X(N-2)-QUOTIENT*X(N-1), Y(N)=Y(N-2)-QUOTIENT*Y(N-1).
540 WHILE REMAINDER<>0
550     DIVIDEND=DIVISOR:DIVISOR=REMAINDER
560     QUOTIENT=FIX(DIVIDEND/DIVISOR)
570     REMAINDER=DIVIDEND-DIVISOR*QUOTIENT
580     IF REMAINDER=0 THEN 610
590     X3=X1-QUOTIENT*X2:Y3=Y1-QUOTIENT*Y2
600     X1=X2:X2=X3:Y1=Y2:Y2=Y3
610 WEND
620 REM CALCULATE BASIC SOLUTION FOR AX + BY = C FROM GCD RESULTS, WHICH HAVE
    PROVIDED AX' + BY' = D BASIC SOLUTION.
630 D=DIVISOR:E=C/D
640 IF C/D<>INT(C/D) THEN PRINT "NO INTEGER SOLUTIONS.":GOTO 760
650 IF SWAP.XY$="YES" THEN SWAP X2,Y2
660 PRINT "The basic solution to the Diophantine equation,"
670 PRINT EQUATION$;" is:"
680 PRINT "X = ";X2*E:PRINT "Y = ";Y2*E
690 PRINT "The GCD of ";A;" and ";B;" is:";ABS(D)
700 PRINT "The parametric equations for all integer answers is:"
710 PRINT "X = ";X2*E;:IF B/D>0 THEN PRINT " +";
720 PRINT B/D;"N, and"
730 PRINT "Y = ";Y2*E;:IF A/D<0 THEN PRINT " +"; ELSE PRINT " -";
740 PRINT ABS(A/D);"N"
750 PRINT "for all integer values N."
760 END

```


April

runkut.doc

TEXT
 "The Runge-Kutta Methods," Benku Thomas.
 April, page 191.

1. RUNKUT - AN INITIAL VALUE ORDINARY DIFFERENTIAL EQUATION SOLVER.

1.1. INTRODUCTION

RUNKUT is a subroutine that contains three Runge-Kutta initial value ordinary differential equation (IVP ODE) solvers. They are:

- A fourth order Fehlberg method.
- A fifth order Verner method.
- An seventh order Verner method.

RUNKUT solves coupled IV ODE's of the form:

$$Y' = F(x,Y)$$

There is no limit to the number of coupled equations in a system to be solved other than the amount of memory available. The differential equations are defined in a user-supplied subroutine--more about that later. The subroutine is called using the following FORTRAN call statement:

```
CALL RUNKUT(XA,Y,XB,NEQN,TOLA,TOLR,HSTART,WORK,
&           IMETH,IERROR,ICOM,FUNC)
```

and must be called from a main "calling" program.

1.2. VARIABLES PASSED TO RUNKUT.

The variables passed are explained below and are listed in the order in which they appear in the call statement. The superscripted numbers refer to notes appearing at the end of this section.

XA (Real-Input)

XA is the starting point of the interval over which integration of the ODE's is to be performed.(1)

Y (Real array of size NEQN - Input/Output)

On entry, Y must contain the initial values of the Y-variables--that is Y must contain the solutions at the point XA. On exit from RUNKUT, Y will contain the solutions at the end of the specified interval--that is at XB.(2)

XB (Real-Input)

XB specifies the end of the interval over which integration is to be performed--the point at which solutions to the differential equations are desired.(1,2)

NEQN (Integer-Input)

NEQN specifies the number of coupled differential equations in the system to be solved.

TOLA (Real-Input)

TOLA is the absolute error tolerance required on the solution.(3)

(continued)

TOLR (Real-Input)

TOLR is the relative error tolerance required on the solution.(3)

HSTART (Real-Input/Output)

On the very first call to RUNKUT in a calling sequence, HSTART must contain a non-zero value as an initial estimate for the step-size.(1) The actual value used is relatively unimportant as the program will optimize step-sizes to keep the tolerances within specified limits and keep the number of function evaluations to a minimum. It is left to the user's discretion to choose a value that will not require excessive readjustment by the program. On exit from RUNKUT, this variable will contain the last step size used by the program. It is recommended that for subsequent calls to RUNKUT, the step size returned by the program be used as the value for HSTART to solve the equations over the immediately following interval.

WORK (Real array of minimum size NEQN*17 - Input)

This is a work area made available to RUNKUT by the calling program.

IMETH (Integer-Input)

Indicates to the solver, the order of the method to be used. IMETH can take on the following values:

1. - Fourth order Fehlberg method
2. - fifth order Verner method
3. - seventh order Verner method

While solving a given set of differential equations it is possible to change the order of the method over successive intervals simply by setting IMETH to the desired value and resetting ICOM(1) to 0 on each subsequent call to RUNKUT in which a change of order is required.(4)

IERROR (Integer-Output)

IERROR is an error status indicator returned by RUNKUT after each call. It can have one of the following values:

- 0 - no error.
- 1 - the sum of TOLA and TOLR is less than 100 times machine epsilon. This can be caused by setting both TOLA and TOLR to 0, or by specifying values that are too small for the program to handle successfully. In this case TOLR is set to a default value.
- 2 - the problem is too stiff for the method to handle, or there is a discontinuity in the function. It is impossible to proceed in either case.(2)
- 3 - both conditions 1 and 2 above have occurred.

IERROR is set to 0 on every entry to RUNKUT.

ICOM (Integer array of size 4)

ICOM is a communications vector. Each array element is used to pass a specific item of information (as defined below) to and from the subroutine.

ICOM(1) (Input)

ICOM(1) must be set to 0 on the first call to RUNKUT. ICOM(1) is internally set to a non-zero value by the program after the first call. For subsequent calls to RUNKUT with the same set of equations, ICOM(1) must be left at this non-zero value. It is possible to change the order of the method used (see also IMETH) over successive intervals. In this case ICOM(1) must be reset to 0 each time the order of the method is changed.(4)

ICOM(2) (Input)

A flag that indicates to the program whether to perform checks on the specified values of TOLA and TOLR.

- 0 - no checking of the error tolerances.
- 1 - program checks if the sum of TOLA and TOLR is at least 100 times machine epsilon. If not, TOLR is set to a default value and the error indicator IERROR is set to 1.

If ICOM(2) is set to 0, the program assumes that it will be receiving non-zero values of at least one of TOLA or TOLR. Failure to check this before entry to RUNKUT could result in fatal program errors. Also, as will be shown in an example, decreasing the error tolerances does not always result in improved solution accuracy.

ICOM(3) (Input)

A flag that indicates to the program whether to use the default values of minimum and maximum step size set by the subroutine.(5)

- 0 - default values of the maximum and minimum step size are used.
- 1 - allows the user to set values for the maximum and minimum permissible step size. These values are set by including the following FORTRAN common block statement in the main "calling" program:

```
COMMON /CONS/HMIN,HMAX
```

ICOM(4) (Output)

Status flag that indicates the presence of round-off error in the solution.

- 0 - No round-off error.
- 1 - Severe round-off error possible in answer.

ICOM(4) is reset to 0 on entry to RUNKUT.

FUNC (Function name)

FUNC is replaced by the name of the subroutine that is part of the main user routine in which the differential equations are specified (See section on user supplied routines). The subroutine name must be declared external by the following FORTRAN statement:

```
EXTERNAL [SUBROUTINE NAME].
```

1.2.1. NOTES

1. XA could be greater than XB. In other words it is possible to use RUNKUT to solve the differential equations in a negative X-direction given initial values at XA. In that case the step-size will also be negative. HSTART could be specified as a negative value, but that is not essential as the program internally adjusts the algebraic sign of the step-size.
2. If a discontinuity or extreme stiffness is encountered, RUNKUT will return in XB, the value of X closest to the point of discontinuity or stiffness, at which the solutions are still within the specified error bounds. The array Y will then contain solutions at this value of X.
3. If the sum of TOLA and TOLR is less than 100 times machine epsilon (for example if neither were defined in the main program), TOLR is set to a default value of $10(6)$ times machine epsilon. TOLA is not set to any default value. The program uses a combination of TOLA and TOLR to determine the accuracies of the solutions, and specifying TOLR to a certain value almost always gives better results than specifying TOLA to the same value.
4. Changing the order of the method on every subsequent call to RUNKUT with a given system of equations can lead to excessive computation times since a large number of constants are re-evaluated each time the order is changed.
5. The default values used by the subroutine are $10(-8)$ for HMIN, and for HMAX as follows:
 - 0.5 - if the fourth order method is used.
 - 1.0 - if the fifth order method is used.
 - 2.0 - if the seventh order method is used.

(continued)

However, if any of these values are larger than the absolute value of the interval width, HMAX is set to the interval width. It is sometimes necessary to specify HMAX to a small value to keep the step sizes to relatively small values that keep the runge-kutta methods within regions of stability. The subroutine detects the onset of problem "stiffness" or the occurrence of a discontinuity in the function by checking if the step size is smaller than HMIN. If the equations are being solved in the negative X-direction, HMIN and HMAX are the smallest and largest permissible step sizes in the negative X-direction respectively. In other words, HMIN and HMAX always define absolute constraints on the step size.

6. Machine epsilon on the IBM PC is very much a function of the compiler used (even with an 8087 math coprocessor installed). For example Microsoft's Fortran-77 compiler (with 8087 support) generates an epsilon of about $10(-16)$, whereas IBM/Ryan-McFarland's Professional Fortran-77 generates an epsilon of about $10(-20)$

1.3. THE USER-SUPPLIED ROUTINES

Two routines must be supplied:

1. A main "calling" program that calls RUNKUT. It must also define the initial values of Y at XA, set TOLA and TOLR, set ICOM(1) to 0 on the first call to RUNKUT, define the number of equations in the system to be solved, dimension a WORK array for RUNKUT, and check for the error status as passed back from RUNKUT through IERROR. See the examples at the end of this section.
2. A subroutine containing the system of differential equations. The subroutine name must be declared EXTERNAL in the main calling program and its name passed to RUNKUT through the parameter FUNC-see above. The subroutine is defined using the following FORTRAN statement:

```
SUBROUTINE [FUNC] (X,Y,YPRIME,NEQN)
```

where the parameters have the following definitions:

X (Real)

Contains the current value of X as set in RUNKUT. Do not alter this value within [func]

Y (Real - array of size NEQN)

Contains the current solutions at X. Do not alter in [func].

YPRIME (Real - array of size NEQN)

YPRIME is set in the subroutine to the differentials dy/dx as follows:

```
YPRIME(1) = F/1/(X,Y(1),...,Y(NEQN))
```

```
.
```

```
YPRIME(NEQN)=F/NEQN/(X,Y(1),...,Y(NEQN))
```

where F/i/ is the i'th differential function.

EQUATION 1. RUNKUT - AN INITIAL VALUE ORDINARY DIFFERENTIAL SOLVER.

1.1. INTRODUCTION

RUNKUT is a subroutine that contains three Runge-Kutta initial value ordinary differential equation (IVP ODE) solvers They are:

- A fourth order Fehlberg method. -
 A fifth order Verner method. - An seventh order
 Verner method.

RUNKUT solves coupled IV ODE's of the form:

$$Y' = F(x, Y)$$

There is no limit to the number of coupled equations in a system to be solved other than the amount of memory available. The differential equations are defined in a user-supplied subroutine--more about that later. The subroutine is called using the following FORTRAN call statement:

```
CALL
RUNKUT(XA,Y,XB,NEQN,TOLA,TOLR,HSTART,WORK,      &
IMETH,IERROR,ICOM,FUNC)
```

and must be called from a main "calling" program.

1.2. VARIABLES PASSED TO RUNKUT.

The variables passed are explained below and are listed in the order in which they appear in the call statement. The superscripted numbers refer to notes appearing at the end of this section.

XA (Real-Input) XA is the starting
 point of the interval over which integration of
 the ODE's is to be performed.(1)

Y (Real array of size NEQN - Input/Output)
 On entry, Y must contain the initial values of the Y-variables--that is Y must contain the solutions at the point XA. On exit from RUNKUT, Y will contain the solutions at the end of the specified interval--that is at XB.(2)

XB (Real-Input) XB specifies the end
 of the interval over which integration is to be
 performed--the point at which solutions to the
 differential equations are desired.(1,2)

NEQN (Integer-Input) NEQN specifies
 the number of coupled differential equations in
 the system to be solved.

TOLA (Real-Input) TOLA is the
 absolute error tolerance required on the
 solution.(3)

TOLR (Real-Input) TOLR is the
 relative error tolerance required on the
 solution.(3)

HSTART (Real-Input/Output) On the
 very first call to RUNKUT in a calling sequence,
 HSTART must contain a non-zero value as an initial estimate
 for the step-size.(1) The actual value used is relatively
 unimportant as the program will optimize step-sizes to keep
 the tolerances within specified limits and keep the number of
 function evaluations to a minimum. It is left to the user's
 discretion to choose a value that will not require excessive
 readjustment by the program. On exit from RUNKUT, this
 variable will contain the last step size used by the program.
 It is recommended that for subsequent calls to RUNKUT, the
 step size returned by the program be used as the value for
 HSTART to solve the equations over the immediately following
 interval.

WORK (Real array of minimum size NEQN*17 - Input)
 This is a work area made available to RUNKUT by the calling
 program.

IMETH (Integer-Input) Indicates to
 the solver, the order of the method to be used.
 IMETH can take on the following values:

(continued)

1. - Fourth order Fehlberg method
2. - fifth order Verner method
3. - seventh order Verner method

While solving a given set of differential equations it is possible to change the order of the method over successive intervals simply by setting IMETH to the desired value and resetting ICOM(1) to 0 on each subsequent call to RUNKUT in which a change of order is required.(4)

IERROR (Integer-Output) IERROR is an error status indicator returned by RUNKUT after each call. It can have one of the following values:

- 0 - no error.
- 1 - the sum of TOLA and TOLR is less than 100 times machine epsilon. This can be caused by setting both TOLA and TOLR to 0, or by specifying values that are too small for the program to handle successfully. In this case TOLR is set to a default value.
- 2 - the problem is too stiff for the method to handle, or there is a discontinuity in the function. It is impossible to proceed in either case.(2)
- 3 - both conditions 1 and 2 above have occurred.

IERROR is set to 0 on every entry to RUNKUT.

ICOM (Integer array of size 4) ICOM is a communications vector. Each array element is used to pass a specific item of information (as defined below) to and from the subroutine.

ICOM(1) (Input) ICOM(1) must be set to 0 on the first call to RUNKUT. ICOM(1) is internally set to a non-zero value by the program after the first call. For subsequent calls to RUNKUT with the same set of equations, ICOM(1) must be left at this non-zero value. It is possible to change the order of the method used (see also IMETH) over successive intervals. In this case ICOM(1) must be reset to 0 each time the order of the method is changed.(4)

ICOM(2) (Input) A flag that indicates to the program whether to perform checks on the specified values of TOLA and TOLR.

- 0 - no checking of the error tolerances.
- 1 - program checks if the sum of TOLA and TOLR is at least 100 times machine epsilon. If not, TOLR is set to a default value and the error indicator IERROR is set to 1.

If ICOM(2) is set to 0, the program assumes that it will be receiving non-zero values of at least one of TOLA or TOLR. Failure to check this before entry to RUNKUT could result in fatal program errors. Also, as will be shown in an example, decreasing the error tolerances does not always result in improved solution accuracy.

ICOM(3) (Input) A flag that indicates to the program whether to use the default values of minimum and maximum step size set by the subroutine.(5)

- 0 - default values of the maximum and minimum step size are used.
- 1 - allows the user to set values for the maximum and minimum permissible step size. These values are set by including the following FORTRAN common block statement in the main "calling" program:

COMMON /CONS/HMIN,HMAX
 ICOM(4) (Output) Status flag
 that indicates the presence of round-off
 error in the solution. 0 - No round-off
 error. 1 - Severe round-off error possible
 in answer.

ICOM(4) is reset to 0 on entry to RUNKUT.

FUNC (Function name) FUNC is
 replaced by the name of the subroutine that is
 part of the main user routine in which the differential
 equations are specified (See section on user supplied
 routines). The subroutine name must be declared external
 by the following FORTRAN statement:

EXTERNAL [SUBROUTINE NAME].

1.2.1. NOTES

1. XA could be greater than XB. In other words
 it is possible to use RUNKUT to solve the
 differential equations in a negative
 X-direction given initial values at XA. In
 that case the step-size will also be negative.
 HSTART could be specified as a negative value, but that
 is not essential as the program internally adjusts the
 algebraic sign of the step-size.

2. If a discontinuity or extreme stiffness is
 encountered, RUNKUT will return in XB, the
 value of X closest to the point of
 discontinuity or stiffness, at which the
 solutions are still within the specified error bounds.
 The array Y will then contain solutions at this value of
 X.

3. If the sum of TOLA and TOLR is less than
 100 times machine epsilon (for example if
 neither were defined in the main program),
 TOLR is set to a default value of 10(6)
 times machine epsilon. TOLA is not set to any
 default value. The program uses a combination of TOLA and
 TOLR to determine the accuracies of the solutions, and
 specifying TOLR to a certain value almost always gives
 better results than specifying TOLA to the same value.

4. Changing the order of the method on every
 subsequent call to RUNKUT with a given
 system of equations can lead to excessive
 computation times since a large number of
 constants are re-evaluated each time the order is
 changed.

5. The default values used by the subroutine
 are 10(-8) for HMIN, and for HMAX as
 follows:

0.5 - if the fourth order method is used.

1.0 - if the fifth order method is used.

2.0 - if the seventh order method is used.

However, if any of these values are larger
 than the absolute value of the interval
 width, HMAX is set to the interval width.
 It is sometimes necessary to specify HMAX
 to a small value to keep the step sizes to relatively
 small values that keep the runge-kutta methods within
 regions of stability. The subroutine detects the onset of
 problem "stiffness" or the occurrence of a discontinuity
 in the function by checking if the step size is smaller
 than HMIN. If the equations are being solved in the
 negative X-direction, HMIN and HMAX are the smallest and
 largest permissible step sizes in the negative
 X-direction respectively. In other words, HMIN and HMAX
 always define absolute constraints on the step size.

(continued)

6. Machine epsilon on the IBM PC is very much a function of the compiler used (even with an 8087 math coprocessor installed). For example Microsoft's Fortran-77 compiler (with 8087 support) generates an epsilon of about $10(-16)$, whereas IBM/Ryan-McFarland's Professional Fortran-77 generates an epsilon of about $10(-20)$.

1.3. THE USER-SUPPLIED ROUTINES

Two routines must be supplied:

1. A main "calling" program that calls RUNKUT. It must also define the initial values of Y at XA, set TOLA and TOLR, set ICOM(1) to 0 on the first call to RUNKUT, define the number of equations in the system to be solved,

dimension a WORK array for RUNKUT, and check for the error status as passed back from RUNKUT through IERROR. See the examples at the end of this section.

2. A subroutine containing the system of differential equations. The subroutine name must be declared EXTERNAL in the main calling program and its name passed to RUNKUT through the parameter FUNC—see above. The subroutine is defined using the following FORTRAN statement:

```
SUBROUTINE [FUNC] (X,Y,YPRIME,NEQN)
```

where the parameters have the following definitions:

X (Real) Contains the current value of X as set in RUNKUT. Do not alter this value within [func]

Y (Real - array of size NEQN) Contains the current solutions at X. Do not alter in [func].

YPRIME (Real - array of size NEQN) YPRIME is set in the subroutine to the differentials dy/dx as follows:

$$YPRIME(1) = F(1/(X,Y(1),\dots,Y(NEQN)))$$

$$YPRIME(NEQN) = F(NEQN/(X,Y(1),\dots,Y(NEQN)))$$

where $F/i/$ is the i 'th differential function.

The following table should be printed out in 132-column format.

TABLE 2 - The numerical solution of the equations in example 3.
(TOLR = $1.E-10$, TOLA = 0.0)

RKF-4			RKV-5		RKV-7	
x	y(1)	y(2)	y(1)	y(2)	y(1)	y(2)
.0	.750000000	.000000000	.750000000	.000000000	.750000000	.000000000
.5	.619768033	.477791373	.619768033	.477791373	.619768033	.477791373
1.0	.294417538	.812178519	.294417538	.812178519	.294417538	.812178519
1.5	-.105176381	.958038093	-.105176381	.958038093	-.105176381	.958038093
2.0	-.490299792	.939874997	-.490299792	.939874997	-.490299792	.939874997
2.5	-.813942831	.799590803	-.813942831	.799590803	-.813942831	.799590803
3.0	-1.054031516	.575706079	-1.054031516	.575706079	-1.054031516	.575706079
3.5	-1.200735041	.300160709	-1.200735041	.300160709	-1.200735041	.300160709
4.0	-1.250000000	.000000000	-1.250000000	.000000000	-1.250000000	.000000000


```

4.5-1.200735041-.300160709 -1.200735041 -.300160709 -1.200735041 -.300160708
5.0-1.054031516-.575706079 -1.054031516 -.575706079 -1.054031516 -.575706078
5.5-.813942831 -.799590803 -.813942831 -.799590803 -.813942831 -.799590803
6.0-.490299792 -.939874997 -.490299792 -.939874997 -.490299792 -.939874997
6.5-.105176381 -.958038093 -.105176381 -.958038093 -.105176381 -.958038093
7.0 .294417538 -.812178519 .294417538 -.812178519 .294417538 -.812178519
7.5 .619768033 -.477791373 .619768033 -.477791373 .619768033 -.477791373
8.0 .750000000 .000000000 .750000000 .000000000 .750000000 .000000000
8.5 .619768032 .477791374 .619768033 .477791373 .619768033 .477791373
9.0 .294417538 .812178520 .294417538 .812178519 .294417539 .812178519
9.5-.105176382 .958038093 -.105176381 .958038093 -.105176381 .958038093
10.0-.490299793 .939874996 -.490299792 .939874997 -.490299792 .939874997
10.5-.813942832 .799590803 -.813942831 .799590803 -.813942831 .799590804
11.0 -1.054031516 .575706078 -1.054031516 .575706079 -1.054031516 .575706079
11.5 -1.200735042 .300160708 -1.200735041 .300160709 -1.200735041 .300160709
12.0 -1.250000000 -.000000001 -1.250000000 .000000000 -1.250000000 .000000001

```

Relative
error 1.220E-010 5.665E-010 6.948E-012 1.184E-010 1.919E-010 8.529E-010
at x=12

Total function evaluations	2928	2640	1872
-------------------------------	------	------	------

The following table should be printed out in 132-column format.

TABLE 3 - Comparison of code accuracy and efficiency for different expressions for the parameter POWER in the step size adjuster.

	RKF-4		RKV-5		RKV-7	
example:	1 (at x=1)	2 (at x=12) y(1)	1 (at x=1)	2 (at x=12)	1 (at x=1)	2 (at x=12)
with POWER = (1/p+1)	[.567E-08] (3216)	[1.22E-10] (2928)	[.138E-08] (2008)	[6.95E-12] (2640)	[.276E-08] (1053)	[1.92E-10] (1872)
with POWER = (1/p)	[.713E-08] (3072)	[1.54E-10] (2844)	[.165E-08] (1968)	[8.68E-12] (2472)	[.316E-08] (1053)	[1.92E-10] (1703)

- The figures in square brackets [] are absolute values of the relative global errors
- The figures in round brackets () are total number of function evaluations taken to reach specified x value from the starting point.
(TOLA = 0, TOLR = 1.E-09 for example 1 and 1.E-10 for example 2)

predict.for

TEXT
"The Runge-Kutta Methods," Benku Thomas.
April, page 191.

```

$NOFLOATCALLS
$NODEBUG
$STORAGE:2

```

C*****

```
subroutine rkp(x,ys,h,neqn,imeth,kt,est,yold,ynew,w,func)
```

```

implicit double precision (a-h,k,p-z)
external func
dimension w(13,neqn),kt(neqn),ys(neqn),est(neqn)
dimension al(12),b(12,12),yold(neqn),ynew(neqn)

```

(continued)

```

common /coeffs/a1,b,a1,a3,a4,a5,a6,a7,a8,a9,a10,a11,
& a12,a13,er1,er3,er4,er5,er6,er7,er8,
& er9,er10,er11,er12,er13,inum
xs=x
do 5 i=1,neqn
  ys(i)=yold(i)
5  continue
  call func(xs,ys,kt,neqn)
  do 7 i=1,neqn
    w(1,i)=kt(i)
7  continue

  do 20 j=2,inum
    j1=j-1
    xs=x+h*a1(j1)
    do 15 i=1,neqn
      ksum=0.d0
      do 10 m=1,j1
        ksum=ksum+b(m,j1)*w(m,i)
10      continue
      ys(i)=yold(i)+h*ksum
15      continue
      call func(xs,ys,kt,neqn)
      do 17 i=1,neqn
        w(j,i)=kt(i)
17      continue
20    continue

c**** evaluate YNEW[i] and the error estimates EST[i]

  if(imeth.EQ.1) then
    do 30 i=1,neqn
      ynew(i)=yold(i)+h*(a1*w(1,i)+a3*w(3,i)+a4*w(4,i)
& +a5*w(5,i)+a6*w(6,i))
      est(i)=h*(er1*w(1,i)+er3*w(3,i)+er4*w(4,i)+er5*w(5,i)
& +er6*w(6,i))
30    continue
  end if

  if(imeth.EQ.2) then
    do 40 i=1,neqn
      ynew(i)=yold(i)+h*(a1*w(1,i)+a3*w(3,i)+a4*w(4,i)
& +a5*w(5,i)+a7*w(7,i)
& +a8*w(8,i))
      est(i)=h*(er1*w(1,i)+er3*w(3,i)+er4*w(4,i)+er5*w(5,i)
& +er6*w(6,i)+er7*w(7,i)+er8*w(8,i))
40    continue
  end if

  if(imeth.EQ.3) then
    do 50 i=1,neqn
      ynew(i)=yold(i)+h*(a1*w(1,i)+a6*w(6,i)+a7*w(7,i)
& +a8*w(8,i)+a9*w(9,i)
& +a12*w(12,i)+a13*w(13,i))
      est(i)=h*(er1*w(1,i)
& +er6*w(6,i)+er7*w(7,i)+er8*w(8,i)+er9*w(9,i)
& +er10*w(10,i)+er11*w(11,i)+er12*w(12,i)
& +er13*w(13,i))
50    continue
  end if

  return
end

```



```
runkut.for
```

```
TEXT
```

```
"The Runge-Kutta Methods," Benku Thomas.  
April, page 191.
```

```
$NOFLOATCALLS
```

```
$NODEBUG
```

```
$STORAGE:2
```

```
C*****
```

```
C    NOTE: Some of the variables are not the same as in the article
```

```
C    For example hadj = v, est=e(i+1), hfact=s, hlim = vlim.
```

```
C*****
```

```
C  
C  
C    subroutine runkut(xa,ya,xb,neqn,tola,tolr,hstart,w,  
C    &                imeth,ierror,icom,func)
```

```
C  
C**** This is simply a front-end subroutine to split up the work  
C**** array specified by the user into smaller arrays
```

```
C  
C    implicit double precision (a-h,k,o-z)  
C    external func  
C    dimension icom(4),w(*)  
C    common /epsil/eps  
  
C    call solver(xa,ya,xb,neqn,tola,tolr,hstart,ierror,imeth,  
C    &          icom(1),icom(2),icom(3),icom(4),w(1),w(1+neqn),  
C    &          w(1+neqn*2),w(1+neqn*3),w(1+neqn*4),func)  
  
C    return  
C    end
```

```
C*****
```

```
C  
C  
C    subroutine solver(xa,ya,xb,neqn,tola,tolr,hstart,ierror,imeth,  
C    &                ist,ichk,idef,iround,kt,est,yold,ynew,w,func)  
  
C    implicit double precision (a-h,k,o-z)  
C    logical hflag  
C    external func  
C    dimension ya(neqn),w(13,neqn),kt(neqn)  
C    dimension est(neqn),yold(neqn),ynew(neqn)  
C    common /cons/hmin,hmaxl,hfact,hlim,power  
C    common /epsil/eps
```

```
C**** If TOLA is set to zero, then a relative error test  
C**** If TOLR is set to zero, then an absolute error test  
C**** If neither are set to zero, then a mixed error test  
C**** If this is the first call to RUNKUT calculate the machine  
C**** epsilon and work out the constants depending on method used.  
C**** To ensure that the results will always be "safe", the value  
C**** EPS used by RUNKUT is actually 20 times the machine epsilon.
```

```
    if(ist.EQ.0) then  
        call caleps  
        call const(ist,imeth,idef)  
        eps=eps*20.d0  
    end if
```

```
C**** Check if the sum of TOLA and TOLR is greater than 10*EPS  
C**** If not, set TOLR to one million times EPS.
```

```
    ierror=0  
    if(ichk.GT.0)then  
        if(tola+tolr.LT.10.d0*eps)then  
            ierror=1  
            tolr=1.d06*eps  
        end if  
    end if  
    iround=0
```

```
    isig=dint((xb-xa)/dabs(xb-xa))  
    hold=hstart  
    x=xa
```

(continued)

```

do 10 i=1,neqn
  yold(i)=ya(i)
10  continue

c**** Set HMAX to the maximum value HMAXL set in CONST unless this
c**** is greater than the interval width, in which case HMAX is
c**** equal to the interval width.

hmax=dmin1(hmaxl,dabs(xa-xb))

c**** call the runge-kutta solver using the current step size
c**** to predict y(n+1)

15  call rkp(x,ya,hold,neqn,imeth,kt,est,yold,ynew,w,func)

c**** calculate step size adjustment factor HADJ
c**** then calculate HNEW using the smallest value of HADJ

hadj=9999.d0
do 20 i=1,neqn
  absest=dabs(est(i))
  if(absest.EQ.0)then
    hadjl=hlím
  else
    if(yold(i).EQ.0)then
      hadjl=((tol+tolr*tolr)/absest)**power
    else
      hadjl=((tol+tolr*dabs(yold(i)))/absest)**power
    end if
  end if
  if(hadjl.LT.hadj) hadj=hadjl
20  continue

c**** adjust the step size to HNEW using the calculated value of HADJ
c**** unless HADJ is greater than HLIM, in which case use HLIM
c**** If HNEW is larger than HMAX, choose HMAX as the new step size
c**** If HNEW is less than HOLD/10. keep HNEW as HOLD/10 to avoid
c**** excessively large swings in the step size

hold1=dabs(hold)
hnew = dmax1(hold1/10.,
&      dmin1(hfact*hold1*(dmin1(hadj,hlím)),hmax))

c**** Check if HOLD is large enough compared to EPS to avoid
c**** severe round-off errors. If HNEW is less than HMIN, the
c**** problem has got too stiff or is discontinuous-exit saving
c**** the last points calculated. If the last step was successful
c**** let YOLD=YNEW and calculate YNEW using new step size. If it
c**** was unsuccessful, recalculate YNEW using reduced step size.
c**** If the HOLD was a reduced step size, then restrict HNEW to HOLD
c**** If XB will be reached or exceeded, exit after calculating
c**** Y at XB.

if(dabs(x).GT.0) then
  if((hnew/(dabs(x)*18.d0)).LE.eps) iround=1
end if
if(hnew.GE.hmin) then
  hnew=isig*hnew
  if(hadj.GE.1.d0) then
    if(isig*(x+hold-xb).LT.0.d0) then
      x=x+hold
      if(.not.hflag) hnew=hold
      hflag=.true.
      do 30 i=1,neqn
        yold(i)=ynew(i)
30      continue
      hold=hnew
      goto 15
    else
      hstart=hnew
      hold=xb-x
      call rkp(x,ya,hold,neqn,imeth,kt,est,yold,ynew,w,func)
      do 50 i=1,neqn
        ya(i)=ynew(i)
50      continue
      end if
    else

```



```

        hflag=.false.
        hold=hnew
        goto 15
    end if
else
    hnew=isig*hnew
    ierror=ierror+2
    xb=x
    do 60 i=1,neqn
        ya(i)=yold(i)
60      continue
    end if
    return
end

```

```

orbit.for

```

```

TEXT
"The Runge-Kutta Methods," Benku Thomas.
April, page 191.

```

```

$NOFLOATCALLS
$NODEBUG
$STORAGE:2
c**** User's calling program
c**** NEQN=4 so dimension of work array = 4*17 = 68

implicit double precision (a-h,o-z)
dimension y(4),work(68),icom(4)
external orbit
common alfasq

open(2,file=' ',status='new')
icom(1)=0
write(*,*) 'imeth=tola=tolr='
read(*,*) imeth,tola,tolr
ecc=0.25d0
alfa=3.141592653589d0/4.d0
alfasq=alfa*alfa
neqn=4
hstart=0.01d0
y(1)=1.d0-ecc
y(2)=0.d0
y(3)=0.d0
y(4)=alfa*dsqrt((1.d0+ecc)/(1.d0-ecc))
x0=0.d0
xb=0.d0
icom(2)=0
icom(3)=0
do 20 j=1,24
    xa=xb
    xb=0.5d0*db1e(j)+x0
    write(2,100)xa,y(1),y(2)
    call runkut(xa,y,xb,neqn,tola,tolr,hstart,work,
&              imeth,ierror,icom,orbit)
    if(ierror.GT.1) then
        write(2,100)xb,y(1),y(2)
        write(2,*) ' ERROR-Problem too stiff or is discontinous'
        close(2)
        stop
    end if
    continue
20  if(icom(4).GT.0)write(2,*) ' Severe round-off error possible'
    stop
100 format(F5.1,3F15.9)
end

c*****
c**** User supplied subroutine that contains the system of
c**** differential equations to be solved.
c**** Notice that in this routine it is necessary to have an array
c**** yprime(neqn)

```

```

subroutine orbit (x,y,yprime,neqn)
implicit double precision (a-h,o-z)
dimension y(neqn),yprime(neqn)
common alfasq

```

```

r=y(1)*y(1)+y(2)*y(2)
r=r*dsqrt(r)/alfasq
yprime(1)=y(3)
yprime(2)=y(4)
yprime(3)=-y(1)/r
yprime(4)=-y(2)/r
return
end

```

```

react.for

```

```

TEXT
"The Runge-Kutta Methods," Benku Thomas.
April, page 191.

```

```

$NOFLOATCALLS

```

```

$NODEBUG

```

```

$STORAGE:2

```

```

C*****

```

```

implicit double precision (a-h,p-z)
dimension y(2),work(34),icom(4)
external freact
common /reacts/ifeval,Da,delta,beta,Hw,Tw

```

```

open(2,file= ' ',status= 'new')
ifeval=0
icom(1)=0
icom(2)=0
icom(3)=0
write(*,*) 'Wall Temp.=, Reactant inlet Temp=, htc='
read(*,*) Tw,Tr,U
Tw=Tw/Tr
U=U/1000.0
write(*,*) ' imeth=, tola=, tolr='
read(*,*) imeth,tola,tolr

```

```

C**** evaluate constants in the equations
Da=2.d0*5.d0/3.d0
beta=0.03d0*1.d04/1.2d0/1.d0/Tr
delta=1.d3/8.3144d0/Tr
Hw=2.d0*U*2.d0/0.1d0/1.2d0/1.d0/3.d0

```

```

C****

```

```

hstart=0.01d0
neqn=2
x0=0.d0
xb=0.d0
y(1)=1.d0
y(2)=1.d0
conc=y(1)*0.03
temp=y(2)*Tr
write(2,30)xa,y(1),y(2),hstart
do 20 j=1,10
  xa=xb
  xb=0.1*dble(j)+x0
  call runkut(xa,y,xb,neqn,tola,tolr,hstart,work,
&          imeth,ierror,icom,freact)

  conc=y(1)*0.03
  temp=y(2)*Tr
  if(ierror.GT.1)then
    write(2,30)xb,y(1),y(2),hstart
    write(2,*)' ERROR-Problem too stiff or is discontinuous'
    close(2)
    stop
  else
    write(2,30)xb,y(1),y(2),hstart
  end if

```



```

20      continue
      if(icom(4).GT.0) write(2,*) ' Severe round-off error possible'
      write(2,*) ' Number of function evaluations = ',ifeval
      close (2)
      stop
30      format(1x,f11.2,1x,d15.6,1x,f15.4,1x,d15.6)
      end
C*****
c      user supplied subroutine containing the system of first
c      order ordinary initial value differential equations

      subroutine freact(x,y,yprime,neqn)

      implicit double precision (a-h,p-z)
      dimension y(neqn),yprime(neqn)
      common /reacts/ifeval,Da,delta,beta,Hw,Tw

      yprime(1)= -Da*y(1)*dexp(delta*(1.d0-1.d0/y(2)))
      yprime(2)= beta*Da*y(1)*dexp(delta*(1.d0-1.d0/y(2)))
&      -Hw*(y(2)-Tw)
      ifeval=ifeval+1

      return
      end

```

```
rkconst.for
```

```

TEXT
"The Runge-Kutta Methods," Benku Thomas.
April, page 191.

```

```

$NOFLOATCALLS
$NODEBUG
$STORAGE:2

```

```

C*****
      subroutine const(ist,imeth,idef)

C****      subroutine supplies constants used in the runge-kutta
C****      method,in the format required by RKP.

      implicit double precision (a-h,p)
      dimension al(12),b(12,12)
      common /coeffs/al,b,a1,a3,a4,a5,a6,a7,a8,a9,a10,a11,
&      a12,a13,er1,er3,er4,er5,er6,er7,er8,
&      er9,er10,er11,er12,er13,inum
      common /cons/hmin,hmax1,hfact,hlim,power

      do 10 j=1,12
      do 5 i=1,12
      b(i,j)=0.d0
5          continue
      al(j)=0.d0
10      continue

      if(imeth.EQ.1) then
C****      Fourth order runge-kutta method specified by
C****      Felberg, E., COMPUTING, 6(1970)p61-71
c      write(*,*) 'Fourth order Runge-Kutta-Fehlberg method'
      al(1)=1.d0/4.d0
      al(2)=3.d0/8.d0
      al(3)=12.d0/13.d0
      al(4)=1.d0
      al(5)=1.d0/2.d0

      b(1,1)=1.d0/4.d0
      b(1,2)=3.d0/32.d0
      b(2,2)=9.d0/32.d0
      b(1,3)=1932.d0/2197.d0
      b(2,3)=-7200.d0/2197.d0
      b(3,3)=7296.d0/2197.d0
      b(1,4)=439.d0/216.d0
      b(2,4)=-8.d0

```

(continued)

```

b(3,4)=3680.d0/513.d0
b(4,4)=-845.d0/4104.d0
b(1,5)=-8.d0/27.d0
b(2,5)=2.d0
b(3,5)=-3544.d0/2565.d0
b(4,5)=1859.d0/4104.d0
b(5,5)=-11.d0/40.d0

```

```

er1=1.d0/360.d0
er3=-384.d0/12825.d0
er4=-41743.d0/1429560.d0
er5=1.d0/50.d0
er6=2.d0/55.d0

```

```

a1=16.d0/135.d0
a2=0.d0
a3=6656.d0/12825.d0
a4=28561.d0/56430.d0
a5=-9.d0/50.d0
a6=2.d0/55.d0

```

```

inum=6
ist=4
if(idef.eq.0) hmaxl=0.5d0
hlml=3.d0
end if

```

```

if(imeth.EQ.2) then
c**** Fifth order runge-kutta method specified by
c**** Verner J.H., SIAM J. Numer. Anal. V15,(1978),p772.(table 5)
c write(*,*) 'Fifth order Runge-Kutta-Verner method'
a1(1)=1.d0/18.d0
a1(2)=1.d0/6.d0
a1(3)=2.d0/9.d0
a1(4)=2.d0/3.d0
a1(5)=1.d0
a1(6)=8.d0/9.d0
a1(7)=1.d0

```

```

b(1,1)=1.d0/18.d0
b(1,2)=-1.d0/12.d0
b(2,2)=1.d0/4.d0
b(1,3)=-2.d0/81.d0
b(2,3)=4.d0/27.d0
b(3,3)=8.d0/81.d0
b(1,4)=40.d0/33.d0
b(2,4)=-4.d0/11.d0
b(3,4)=-56.d0/11.d0
b(4,4)=54.d0/11.d0
b(1,5)=-369.d0/73.d0
b(2,5)=72.d0/73.d0
b(3,5)=5380.d0/219.d0
b(4,5)=-12285.d0/584.d0
b(5,5)=2695.d0/1752.d0
b(1,6)=-8716.d0/891.d0
b(2,6)=656.d0/297.d0
b(3,6)=39520.d0/891.d0
b(4,6)=-416.d0/11.d0
b(5,6)=52.d0/27.d0
b(1,7)=3015.d0/256.d0
b(2,7)=-9.d0/4.d0
b(3,7)=-4219.d0/78.d0
b(4,7)=5985.d0/128.d0
b(5,7)=-539.d0/384.d0
b(7,7)=693.d0/3328.d0

```

```

er1=33.d0/640.d0
er3=-132.d0/325.d0
er4=891.d0/2240.d0
er5=-33.d0/320.d0
er6=-73.d0/700.d0
er7=891.d0/8320.d0
er8=2.d0/35.d0

```



```

a1=57.d0/640.d0
a3=-16.d0/65.d0
a4=1377.d0/2240.d0
a5=121.d0/320.d0
a7=891.d0/8320.d0
a8=2.d0/35.d0

inum=8
ist=5
if(idef.eq.0) hmaxl=1.0d0
hlim1=4.d0
end if

if(imeth.EQ.3) then
c**** Seventh order runge-kutta method specified by
c**** Verner J.H., SIAM J. Numer. Anal. V15,(1978),p772.(table 7)
c write(*,*) 'Seventh order Runge-Kutta-Verner method'
a1(1)=1.d0/4.d0
a1(2)=1.d0/12.d0
a1(3)=1.d0/8.d0
a1(4)=2.d0/5.d0
a1(5)=1.d0/2.d0
a1(6)=6.d0/7.d0
a1(7)=1.d0/7.d0
a1(8)=2.d0/3.d0
a1(9)=2.d0/7.d0
a1(10)=1.d0
a1(11)=1.d0/3.d0
a1(12)=1.d0

b(1,1)=1.d0/4.d0
b(1,2)=5.d0/72.d0
b(2,2)=1.d0/72.d0
b(1,3)=1.d0/32.d0
b(3,3)=3.d0/32.d0
b(1,4)=106.d0/125.d0
b(3,4)=-408.d0/125.d0
b(4,4)=352.d0/125.d0
b(1,5)=1.d0/48.d0
b(4,5)=8.d0/33.d0
b(5,5)=125.d0/528.d0
b(1,6)=-1263.d0/2401.d0
b(4,6)=39936d0/26411d0
b(5,6)=-64125.d0/26411d0
b(6,6)=5520.d0/2401.d0
b(1,7)=37.d0/392.d0
b(5,7)=1625.d0/9408.d0
b(6,7)=-2.d0/15.d0
b(7,7)=61.d0/6720.d0
b(1,8)=17176.d0/25515.d0
b(4,8)=-47104.d0/25515.d0
b(5,8)=1325.d0/504.d0
b(6,8)=-41792.d0/25515.d0
b(7,8)=20237.d0/145800.d0
b(8,8)=4312.d0/6075.d0
b(1,9)=-23834.d0/180075.d0
b(4,9)=-77824.d0/1980825.d0
b(5,9)=-636635.d0/633864.d0
b(6,9)=254048.d0/300125.d0
b(7,9)=-183.d0/7000.d0
b(8,9)=8.d0/11.d0
b(9,9)=-324.d0/3773.d0
b(1,10)=12733.d0/7600.d0
b(4,10)=-20032.d0/5225.d0
b(5,10)=456485.d0/80256.d0
b(6,10)=-42599.d0/7125.d0
b(7,10)=339227.d0/912000d0
b(8,10)=-1029.d0/4180.d0
b(9,10)=1701.d0/1408.d0
b(10,10)=5145.d0/2432.d0
b(1,11)=-27061.d0/204120.d0
b(4,11)=40448.d0/280665.d0
b(5,11)=-1353775.d0/1197504.d0
b(6,11)=17662.d0/25515.d0
b(7,11)=-71687.d0/1166400d0
b(8,11)=98.d0/225.d0

```

(continued)

```

b(9,11)=1.d0/16.d0
b(10,11)=3773.d0/11664.d0
b(1,12)=11203.d0/8680.d0
b(4,12)=-38144.d0/11935.d0
b(5,12)=2354425.d0/458304.d0
b(6,12)=-84046.d0/16275.d0
b(7,12)=673309.d0/1636800.d0
b(8,12)=4704.d0/8525.d0
b(9,12)=9477.d0/10912.d0
b(10,12)=-1029.d0/992.d0
b(12,12)=729.d0/341.d0

```

```

er1=-1.d0/480.d0
er6=-16.d0/375.d0
er7=-2401.d0/528000.d0
er8=2401.d0/132000.d0
er9=243.d0/14080.d0
er10=-2401.d0/19200.d0
er11=-19.d0/450.d0
er12=243.d0/1760.d0
er13=31.d0/720.d0

```

```

a1=31.d0/720.d0
a6=16.d0/75.d0
a7=16807.d0/79200.d0
a8=16807.d0/79200.d0
a9=243.d0/1760.d0
a12=243.d0/1760.d0
a13=31.d0/720.d0

```

```

inum=13
ist=7
if(idef.eq.0) hmaxl=2.d0
hlim1=5.d0
end if

```

```

power=1.d0/dbl(ist+1)
hfact=0.5d0**(1.d0/dbl(ist-1))
hlim=hlim1/hfact
if(idef.eq.0) hmin=1.d-06
c write(*,*) 'Constants evaluated'
return
end

```

epsilon.for

\$NOFLOATCALLS

\$NODEBUG

\$STORAGE:2

C*****

subroutine caleps

```

c**** subroutine calculates the machine epsilon EPS using an
c**** algorithm adapted from Forsythe et. al. "Computer Methods
c**** for Mathematical Computations", Prentice-Hall, N.J. (1977).
c**** The EPS calculated can differ from the true machine EPS by
c**** at most a factor of 2.

```

```

double precision eps
common /epsil/eps

```

```

eps=1.d0
10 eps=.5d0*eps
if((eps+1.d0).GT.1.d0) goto 10
c write(*,*) 'Machine Epsilon used=',eps
return
end

```


therm.for

TEXT
 "The Runge-Kutta Methods," Benku Thomas.
 April, page 191.

```

implicit double precision (a-h,o-z)
dimension y(1), work(17), icom(4)
external thermo
common etherm,ifeval,j

open(2,file=' ',status='new')
ifeval=0
icom(1)=0
icom(2)=0
icom(3)=0
neqn=1
write(*,*) 'etherm=, imeth=, tola=, tolr='
read(*,*) etherm,imeth,tola,tolr
hstart=0.01d0
y(1)=1.d0
x0=0.d0
xb=0.d0
do 20 j=1,6
  xa=xb
  xb=0.2d0*dble(j)+x0
  abserr=dexp(-etherm*xa)-y(1)
  relerr=abserr/dexp(-etherm*xa)
  write(2,100)xa,y(1),abserr,relerr
  call runkut(xa,y,xb,neqn,tola,tolr,hstart,work,
&             imeth,ierror,icom,thermo)
  if(ierror.GT.1) then
    write(2,100)xb,y(1),abserr,relerr
    write(2,*) ' ERROR-Problem too stiff or is discontinuous'
    close(2)
    stop
  end if
20 continue
if(icom(4).GT.0) write(2,*) 'Round-off error possible'
write(2,*) 'Number of function evaluations = ',ifeval
close (2)
stop
100 format(F10.5,4E14.6)
end

C*****
subroutine thermo (x,y,yprime,neqn)
implicit double precision (a-h,o-z)
dimension y(neqn), yprime(neqn)
common etherm,ifeval,j

yprime(1)= -etherm*y(1)
if(j.LE.5) ifeval=ifeval+1
return
end

```

exampler.for

TEXT
 "The Runge-Kutta Methods," Benku Thomas.
 April, page 191.

RUNGE KUTTA EXAMPLES.

Let us look at three examples which we will attempt to solve by using the FORTRAN implementations of the algorithms outlined in the BYTE article.

EXAMPLE 1. This is the radioactivity problem from the article. The dimensionless form of the equation was solved, with the decay constant set to a value of 25. Since the solution produced values close to zero, a relative error tolerance was used, and its value

(continued)

set to $10(-9)$. The calling (user) program is similar to the listing for example 3. The result of running the program with the fifth order method is given in table 1a. For comparison, table 1b gives the absolute and relative global errors at each step using each of the methods. The total number of function evaluations is a measure of the efficiency of the method and the global error, a measure of the accuracy. The expression for the parameter POWER, used in the formula for the step size adjustment factor v , was $1/(p+1)$.

TABLE 1a - Results of solving the equation $y' = -25y$ using the variable step size fifth order method. (Relative error tol. = $1.E-09$)

x	y(x)	Absolute global error	Relative global error
0.0	0.100000E+01	0.000000E+00	0.000000E+00
0.2	0.673795E-02	-0.186541E-11	-0.276851E-09
0.4	0.453999E-04	-0.250863E-13	-0.552562E-09
0.6	0.305902E-06	-0.253371E-15	-0.828274E-09
0.8	0.206115E-08	-0.227547E-17	-0.110398E-08
1.0	0.138879E-10	-0.191612E-19	-0.137970E-08
Total number of function evaluations =		2008	

TABLE 1b - Comparison of the relative global error in the numerical solutions of the equation $y' = -25y$, using the 4th, 5th and 7th order variable step methods. (TOLR = $1.E-09$, TOLA = 0.0)

x	RKF-4	RKV-5	RKV-7
0.0	0.000000E+00	0.000000E+00	0.000000E+00
0.2	0.113247E-08	-0.276851E-09	-0.519695E-09
0.4	0.226672E-08	-0.552562E-09	-0.107925E-08
0.6	0.340097E-08	-0.828274E-09	-0.163881E-08
0.8	0.453523E-08	-0.110398E-08	-0.219837E-08
1.0	0.566949E-08	-0.137970E-08	-0.275792E-08
Total function evaluations	3216	2008	1053

EXAMPLE 2. The classic stiff problem involving the rate of formation of methyl iodide in a nuclear reactor is solved, using a reactant inlet temperature of 800 K. The temperature of the cooling fluid on the outside of the reactor (T/w) was varied from 400 K to 800 K. The results obtained from the numerical solution are plotted in figure 2 for cooling fluid temperatures of 400K, 600K, and 800K. This gives the dimensionless temperature in the reactor as a function of fractional distance down the length of the tubular reactor. From the physics of the problem, we know that the reaction gives off heat, that the amount of heat released will increase with the rate of reaction which in turn is proportional to the amount of reactant present. At the inlet of the reactor there is a large amount of reactant present, so the rate will be high, resulting in a high heat release. The heat is being produced by the reaction faster than can be removed by the cooling fluid, so the mixture heats up, driving the reaction even faster. However, the amount of reactant is also falling down the length of the reactor, so eventually the reaction slows down. The heat released drops off until a point is reached when the amount of heat removed by the cooling fluid just equals the rate of heat production from the reaction. The temperature tops off and then starts to fall as the rate of reaction drops even further. We see

this maximum in the temperature profiles as expected. In addition, if the temperature of the cooling fluid is lowered, the amount of heat removed increases and so the maximum in the temperature will be lower, and will shift towards the inlet. This trend too is observed in the profiles. The rate of reaction is proportional to the temperature, so we would expect to see less reactant in the outlet of the hotter reactor. The (dimensionless) outlet concentrations obtained from the program were .03548, .03424, and .03321 for the coolant temperatures of 400K, 600K, and 800K respectively. Although we do not know the analytical solution to this problem, we are able to say that our program is producing reasonable results by checking to see if the trends are what we would expect.

EXAMPLE 3. This problem describes the orbit of a satellite. The problem statement and the derivation of the differential equations are described in Shampine, L.F., and M.K. Gordon, COMPUTER SOLUTION OF ORDINARY DIFFERENTIAL EQUATIONS: THE INITIAL VALUE PROBLEM, Freeman, San Francisco (1975). For our purposes, we will only deal with the final set of equations. They are included in the FORTRAN listing for the user supplied programs for this problem. The format used for this program is similar to that used for the other examples.

The solutions using the three methods over the range $t=0$ to $t=12$ are given in table 2. A relative error tolerance of $10(-10)$ was used. The values of $y/1/$ and $y/2/$ at $x=12$ should be -1.25 and 0 respectively.

NOTE: TABLES 2 & 3 ARE IN SEPARATE FILES ON THIS DISK. THEY MUST BE PRINTED OUT IN 132-COLUMN FORMAT TO BE READABLE.

Table 3 compares the use of the two expressions for the step size adjustment factor - that is for the parameter $POWER = 1/p$ and $1/(p+1)$. It would appear from this that the expression that contains the term $1/(p+1)$ is preferable. Also, the fifth order method gives the most accurate results and the seventh order method uses the least number of function evaluations (the fastest). However, if memory usage is a consideration, the fourth order method is the code that requires the least amount of memory, and is the easiest to code.

atomcc.exe

BINARY

"The ATOMCC Toolbox," Y. F. Chang.
 April, page 215. If you have an MS-DOS-based system,
 also download cdrdev.obj, crdev.obj, drdev.obj, rdev.obj,
 and manual.doc. For other formats, download atspgm.for
 and painleve.for.

c This program was produced by the ATOMCC translator version 7.10
 c Copyright (C) 1985, Y. F. Chang

c Portions (c) Copyright, Microsoft Corp., 1981. All rights reserved.

c This program was written for the following inputs

c

C FIRST PAINLEVE TRANSCENDENT

C DIFF(Y,T,2) = 6.0*Y*Y + T

c-----

c no instructions in second input block

c-----

```
COMMON /IPASS/ LENSER,LENVAR,MPRINT,MSTIFF,LRUN,
+ KTRDCV,KNTSTP,KTSTIF,KXPNUM,KDIGS,KENDFG,NTERMS,NOPT
A /RPASS/ RADIUS,ERRLIM,ADJSTF,RCREAL,RCIMAG,RDCERR
B /CPASS/ START,END,ORDER
C /DPASS/ H,HNEW,XPRINT,DLTXPT
DIMENSION TMPS( 36, 1)
CHARACTER*6 NAMES
EQUIVALENCE (TMPS(1,1),Y(1))
DIMENSION NAMES(1), Y(36), T(2), TMPAAB(30), TMPAAA(30)
DATA NAMES(1)/'Y.....'/
```

(continued)

```

      Y(33) = 1.1
10  FORMAT(72H  ATOMCC  Ver. 7.10, Copyright (C) 1985, Y. F. Chang; S
      Aolution results./9H  *****)
11  FORMAT(/5X,11HStep number,I6,13H at the point,1P1E12.4/1X,
      A 9Hvalues of )
12  FORMAT(1X, A6,(1X,1P4E13. 5))
13  FORMAT(5X,21HStepsize adjusted to ,1PE13.5)
14  FORMAT(/5X,35HThe solution stopped normally after, I4,24H steps as
      a set by nsteps. )
16  FORMAT(/5X,63HThe adjustment for stepsize seems to be in a loop. P
      Alease try a /5X,22Hshorter series length. )
      WRITE(*,10)
c-----
c Initialize variables to default values.
c-----
      NSTEPS = 40
      H = 1.E0
      ERR LIM = 1.E- 6
      LENSER = 30
      MPRINT = 4
      NTERMS = 2
      KTRDCV = 1
      ADJSTF = 1.E-2
      MSTIFF = 0
      DLTXP T = 0.E0
c-----
c start of third  input block
c-----
C READ INTEGRATION INTERVAL AND INITIAL CONDITIONS.
      READ(5,1010) START,END,Y(1),Y(2)
1010  FORMAT(4F10.3)
      WRITE(*,1020) START,END,Y(1),Y(2)
1020  FORMAT(' SOLVE THE FIRST PAINLEVE TRANSCENDENT' /
      + ' INTERVAL: ',2F10.3 /
      + ' INITIAL CONDITIONS:',2F10.3 /)
c-----
c end of third  input block
c-----
c More initializations
c-----
      DLTXP T = SIGN(DLTXP T,(END-START))
      H = SIGN(H,(END-START))
      KDIGS = 6
      XPRINT = START + DLTXP T
      KXPNUM = 35
      LENVAR = 36
      LRUN = 1
      KTSTIF = 0
      NUMEQS = 1
      IF(LENSER.GT.(LENVAR- 6)) LENSER = LENVAR - 6
      IF(MPRINT.LT.2) GO TO 17
      WRITE(*,11) KTSTIF,START
      K = Y(33)
      WRITE(*,12) NAMES(K),Y(1),Y(2)
c-----
c Loop for integration steps.  Inside the loop, print the desired output
c-----
      17 DO 27 KINTS=1,NSTEPS
          KOUNT = 0
          KNTSTP = KINTS
          CONTINUE

          T(1) = START
          T(2) = H
          Y(2) = Y(2)*(H)
c-----
c Preliminary series calculations
c-----
          TMPAAA(1) = 6.E0*Y(1)
          TMPAAB(1) = TMPAAA(1)*Y(1)
          Y(3) = (TMPAAB(1) + T(1))*(H*H/2.E0)
          TMPAAA(2) = 6.E0*Y(2)
          TMPAAB(2) = TMPAAA(1)*Y(2) + TMPAAA(2)*Y(1)
          Y(4) = (TMPAAB(2) + T(2))*(H*H/6.E0)

```



```

c-----
c Loop for series calculations
c-----
      DO 23 K= 5,LENSER
        KA = K - 1
        KB = K - 2
        TMPAAA(KB) = 6.E0*Y(KB)
        TMPAAB(KB) = 0.E0
        KZ = 1 + KB
        DO 30 N=1, KB
          L = KZ - N
          TMPAAB(KB) = TMPAAB(KB) + TMPAAA(N)*Y(L)
30      CONTINUE
        Y(K) = (TMPAAB(KB))*(H*H/(KB*KA))
c-----
c Test and adjust H to avoid over/under flow.
c-----
      IF(MSTIFF.GE.20 .AND. KTSTIF.GT.0) GO TO 23
      TMP = ABS(Y(K))
      IF(TMP.LE.1.E-35) GO TO 23
      IF(TMP.LT.1.E20 .AND. TMP.GT.1.E-20) GO TO 23
      IF(KTSTIF.NE.0 .AND. TMP.LT.1.0) GO TO 23
      KOUNT = KOUNT + 1
      IF(KOUNT.LT.9) GO TO 22
      WRITE(*,16)
      GO TO 28
22     CONTINUE
      Y(2) = Y(2)/(H)
      H = H * TMP**(0.3/(1-K))
      IF(MPRINT.GE.4) WRITE(*,13) H
      GO TO 19
23     LRUN = 1
c-----
c Calculate radius of convergence and take optimum step.
c-----
      CALL RDCV(TMPS,LENVAR,NUMEQS,NAMES)
24     CALL RSET(TMPS,LENVAR,NUMEQS,NAMES)
c-----
c no instructions in fourth input block
c-----
25     GO TO (26,28,24), KENDFG
26     H = SIGN(RADIUS,H)
      START = START + HNEW
      IF(MPRINT.LT.4) GO TO 27
      WRITE(*,11) KNTSTP, START
      K = Y(33)
      WRITE(*,12) NAMES(K),Y(1), Y(2)
27     CONTINUE
      WRITE(*,14) NSTEPS
28     CONTINUE
29     STOP
      END

```

onebody.bas

TEXT

"The Runge-Kutta Methods" Benku Thomas.
 April, page 191. This is Listing A, page 197, written by David M. Leo.

```

0EM N = THE NUMBER OF INTEGRATORS
20 REM V(I) = INTEGRATOR INPUTS
30 REM X(I) = INTEGRATOR OUTPUTS
40 REM X(1) = VELOCITY; X(2) = DISPLACEMENT
50 REM C = THE DAMPING CONSTANT
60 REM M = MASS; K = SPRING CONSTANT
70 REM TMIN = TIME AT WHICH PRINTING TO OUTPUT FILE BEGINS
80 REM TMAX = TIME AT WHICH PRINTOUT ENDS
90 REM DT = TIME STEP SIZE
100 REM *****
110 OPEN"OUT.DAT" FOR OUTPUT AS #1
120 N=2:C=1500:M=2:K=100000!
130 DIM X(N),K1(N),K2(N),K3(N),K4(N),DUM(N),V(N)
140 DT=.0001:TMIN=DT:TMAX=500*DT
150 REM FNX(T) IS THE DRIVING FORCE

```

(continued)

```

160 DEF FNX(T)=5000*(SIN(200*T))^5
170 REM NEXT ARE THE INITIAL CONDITIONS ON VELOCITY,X(1),AND DISPLACEMENT X(2)
180 X(1)=.001:X(2)=.001
190 REM NEXT IS THE EQUATION FOR THE INPUT TO INTEGRATOR #1
200 REM IN THIS CASE, THERE IS COULOMB DAMPING
210 V(1)=(FNX(T)-K*X(2)-C*SGN(X(1)))/M:V(2)=X(1)
220 REM NEXT ARE THE STEPS IN THE INTEGRATION ACROSS ONE TIME STEP
230 COUNT=COUNT+1:IF COUNT=2 THEN 260:IF COUNT=3 THEN 270:IF COUNT=4 THEN 280
240 FOR I= 1 TO N:K1(I)=DT*V(I):NEXT I
250 T=T+DT/2:FOR I=1 TO N:DUM(I)=X(I):X(I)=DUM(I)+K1(I)/2:NEXT I:IF COUNT=1
THEN 210
260 FOR I= 1 TO N:K2(I)=DT*V(I):X(I)=DUM(I)+K2(I)/2:NEXT I:IF COUNT=2 THEN 210
270 FOR I=1 TO N:K3(I)=DT*V(I):X(I)=DUM(I)+K3(I):NEXT I:T=T+DT/2:IF COUNT=3
THEN 210
280 FOR I=1 TO
N:K4(I)=DT*V(I):X(I)=DUM(I)+(K1(I)+K4(I))/6+(K2(I)+K3(I))/3:NEXT I
290 REM NEXT STEPS APPROXIMATE STATIC/DYNAMIC FRICTION
300 IF ABS(X(1))>.5 THEN 330
310 C=3000
320 GOTO 340
330 C=1500
340 IF X(2)>1! THEN 440
350 REM FROM RUNNING THIS A FEW TIMES, WE EXPECT ABS(DISPLACEMENT) < .030
360 REM OVER THIS RANGE, THE NONLINEAR SPRING RATE CAN BE APPROXIMATED AS
FOLLOWS:
370 K=100000!+300000!*(SIN(50*X(2)))^4
380 IF T<TMIN THEN 210
390 PRINT #1,FNX(T):FOR I=1 TO N:PRINT #1,X(I):NEXT I
400 IF T>TMAX THEN 420
410 GOTO 210
420 CLOSE
430 GOTO 450
440 PRINT" DIVERGENT OSCILLATION ENCOUNTERED"
450 END

```

```
remes.c
```

```
TEXT
```

```
"Computer Approximations," Stephen Moshier. See chbev1.c.
```

```
0 remes.c
```

```

*
* This is an interactive program that computes least maximum polynomial
* and rational approximations.
*/
#define P 15 /* max total degree of polynomials, + 2 */
#define N 20 /* number of items to tabulate for display */
extern double PI; /* 3.14159... */

static int IPS[P] = {0,}; /* simq() work vector */
static double AA[P*P] = {0.0,}; /* coefficient matrix */
static double BB[P] = {0.0,}; /* right hand side vector */
static double param[P] = {0.0,}; /* solution vector */
static double xx[P] = {0.0,}; /* points in approximation interval */
static double ref[N+1] = {0.0,}; /* function values for display */
static int n = 0; /* degree of numerator polynomial */
static int d = 0; /* degree of denominator polynomial */
static int nd1 = 0; /* n + d + 1 */

main()
{
int i, ip, j, ncheb, neq, relerr;
double a, apstrt, apwid, b, c, x, y, z;
char s[40];
double abs(), approx(), cos(), func();
int simq();

printf( "\nRational Approximation by Remes Algorithm\n\n" );

START:

printf( "Relative error (y or n) ? " ); /* Ask for error criterion */
gets( s ); /* read in a line of characters */

```



```

relerr = 0;
if( s[0] == 'y' )
    relerr = 1;

printf( "Degree of numerator polynomial? " );
gets( s ); /* read line */
sscanf( s, "%d", &n ); /* decode characters */

printf( "Degree of denominator polynomial? " );
gets( s );
sscanf( s, "%d", &d );

printf( "Start of approximation interval ? " );
gets( s );
sscanf( s, "%E", &apstrt );

printf( "Width of approximation interval ? " );
gets( s );
sscanf( s, "%E", &apwid );
nd1 = n + d + 1;

/*                                     remes.c                                     2 */

/* Supply initial guesses for points in approximation interval */

if( d == 0 )
{
    /* there is no denominator polynomial */
    neq = n + 2; /* The number of equations to solve */
    ncheb = n + 1; /* Degree of Chebyshev error estimate */
    /* Extrema of Chebyshev polynomial */
    a = ncheb;
    for( i=0; i<neq; i++ )
    {
        xx[i] = apstrt
            + 0.5 * apwid * (1.0 - cos( (PI * i) / a ));
    }
}
else
{
    /* there is a denominator polynomial */
    neq = nd1;
    ncheb = nd1;
    /* Zeros of Chebyshev polynomial */
    a = 2.0 * ncheb;
    for( i=0; i<ncheb; i++ )
    {
        xx[i] = apstrt
            + 0.5 * apwid * (1.0 - cos( PI * (2*i+1) / a ));
    }
    c = 0.0; /* deviation at solution points */
}

/* calculate function table for error curve display */
a = apwid/N;
b = apstrt;
for( i=0; i<=N; i++ )
{
    ref[i] = func(b); /* func is the f(x) to be approximated */
    b += a;
}

/*                                     remes.c                                     3 */

LOOP:

/* Display old values of guesses and let user change them if desired */

/* First do the deviation guess if rational form */
if( d > 0 )
{
    /* there is a denominator */
    c = abs(c); /* deviation at solution points */
    printf( "deviation = %.4E ? ", c );
    gets( s );
    if( s[0] != '\0' ) /* if input is not a null line, */
        sscanf( s, "%E", &c ); /* then decode the number */
}

```

(continued)

```

    }
else      /* no denominator: */
    c = 1.0; /* coefficient of the deviation */

/* Read in guesses for locations of solution */
for( i=0; i<neq; i++ )
{
    printf( "x[%d] = %.4E ? ", i, xx[i] );
    gets( s );
    if( s[0] != '\0' )
    {
        sscanf( s, "%E", &x );
        xx[i] = x;
    }
}

/*
                                remes.c                4 */

/* Set up the equations for solution by simq() */
for( i=0; i<neq; i++ )
{
    ip = neq * i; /* offset to 1st element of this row of matrix */
    x = xx[i]; /* the guess for this row */
    /* right-hand-side vector */
    y = func(x); /* accurate function value f(x) */
    if( d > 0 )
    {
        /* add the deviation if rational form */
        if( relerr ) /* relative error criterion */
            y = y * (1.0+c);
        else /* absolute error criterion */
            y = y + c;
    }
    /* insert powers of x[i] for numerator coefficients */
    z = 1.0;
    for( j=0; j<=n; j++ )
    {
        AA[ip+j] = z;
        z = z * x;
    }
    /* insert denominator terms, if any */
    if( d > 0 )
    {
        z = 1.0;
        for( j=0; j<d; j++ )
        {
            AA[ip+n+1+j] = -y * z;
            z = z * x;
        }
        BB[i] = y * z; /* right hand side vector */
    }
    else
    {
        /* no denominator */
        BB[i] = y; /* right hand side vector */
        y = c;
        if( relerr )
            y = y * BB[i];
        AA[ip+n+1] = y;
    }
    c = -1.0 * c; /* switch sign of deviation for next row */
}

/* Solve the simultaneous linear equations */
simq( &AA[0], &BB[0], &param[0], neq, 0, &IPS[0] );

/*
                                remes.c                5 */

/* Display the results */

j = 0; /* printout variable */
ip = 0; /* solution vector counter */
printf( "Numerator coefficients:\n" );
for( i=0; i<=n; i++, j++, ip++ )
{
    if( j >= 7 )
    {
        printf( "\n" );
    }
}

```



```

        j = 0;
        printf( "%.4E ", param[ip] );
    }
    if( d > 0 )
    {
        j = 0;
        printf( "\nDenominator coefficients:\n" );
        for( i=0; i<d; i++, j++, ip++ )
        {
            if( j >= 7 )
            {
                printf( "\n" );
                j = 0;
            }
            printf( "%.4E ", param[ip] );
        }
    }
    else
        printf( "\nDeviation: %.4E", param[ip] );

    /* Display table of function and approximation error */
    printf( "\n\n      x          func      approx      error\n" );
    a = apwidht/N;
    b = apstrt;
    for( i=0; i<=N; i++ )
    {
        x = b + i * a;
        z = approx(x);
        y = z - ref[i];
        if( relerr && (z != 0.0) )
            y = y/z;

        printf( "%.3E %.3E %.3E %.3E\n", x, ref[i], z, y );
    }
    printf( "Another iteration (y or n)? " ); /* Ask what to do next */
    gets( s );
    if( s[0] == 'y' )
        goto LOOP;
    if( s[0] == 'x' )
        exit(0);
    else
        goto START;
}

/*
                                     remes.c
                                     6 */

/* This subroutine computes the rational form P(x)/Q(x)
 * using coefficients from the solution vector param[]
 */
double approx(x)
double x;
{
    double yn, yd;
    int i;

    yn = param[n]; /* highest order numerator coefficient */
    for( i=n-1; i>=0; i-- ) /* work backwards toward the constant term */
        yn = x * yn + param[i];

    if( d > 0 )
    {
        yd = x + param[n+d]; /* highest order coefficient = 1.0 */
        for( i=n+d-1; i>n; i-- )
            yd = x * yd + param[i];
    }
    else
        yd = 1.0; /* if there is no denominator */

    return( yn/yd );
}

/* Put here an accurate routine for the function
 * to be approximated

```

(continued)

April

```
*/  
double func(x)  
double x;  
{  
double cos(), sqrt();  
double y;  
  
y = sqrt(x);  
return( cos(PI * y / 2.0) );  
}
```

```
Screen #1  
( begin Dragon curve )  
CREATE CURVE ( a FORGETable name)  
  
CARTESIAN OFF  
  
: RECURS SMUDGE ; IMMEDIATE ( trick verb for recursion)  
  
VARIABLE ANGLE  
VARIABLE XCOORD  
VARIABLE YCOORD  
VARIABLE STEPSIZE  
  
: TURN ( deltangle-- | turn sign*delta)  
  ANGLE +! ;  
  
2 4 THRU
```

```
Screen #2  
: MOVE ( -- | takes a step in present turtle direction)  
  STEPSIZE @ DUP  
  ANGLE @ COS * 10000 / ( r* cos of theta) XCOORD @ +  
  DUP ( newX) XCOORD ! ( update X)  
  SWAP  
  ANGLE @ SIN * 10000 / ( r* sine theta ) YCOORD @ +  
  DUP ( newY) YCOORD ! ( update Y)  
  DRAW.TO ;
```

```
Screen #3  
: DRAGON ( sign level-- | )  
  DUP ( level) 0=  
  IF ( at bottom of recursion)  
    DROP ( level) DROP ( sign) MOVE ( by stepsize)  
  ELSE  
    OVER 45 * TURN ( getsign and turn)  
    1 ( newsign)  
    OVER 1- ( level=level-1)  
    RECURS DRAGON RECURS  
  
    OVER -90 * TURN ( getsign & turn)  
    -1 ( newsign) ( edit to +1 for diff curve)  
    OVER 1- ( level=level-1)  
    RECURS DRAGON RECURS  
    DROP ( input level) 45 * TURN ( getsign and turn)  
  THEN ;
```

```
Screen #4  
: DCURVE ( level --| )  
  ( init pen position)  
  PAGE 100 XCOORD ! 90 YCOORD ! 360 6 * ANGLE !  
  WHITE PENPAT XCOORD @ YCOORD @ MOVE.TO  
  
  PEN.NORMAL  
  1 STEPSIZE !  
  1 SWAP ( level) DRAGON  
  
  WHITE PENPAT 4 10 MOVE.TO PEN.NORMAL ;
```


simq.c

TEXT

"Computer Approximations," Stephen Moshier. See chbevl.c.

```

0
*      Solution of simultaneous linear equations  $AX = B$ 
*      by Gaussian elimination with partial pivoting
*
*
*
* SYNOPSIS:
*
* double A[n*n], B[n], X[n];
* int n, flag;
* int IPS[];
* int simq();
*
* ercode = simq( A, B, X, n, flag, IPS );
*
*
* DESCRIPTION:
*
* B, X, IPS are vectors of length n.
* A is an  $n \times n$  matrix (i.e., a vector of length  $n*n$ ),
* stored row-wise: that is,  $A(i,j) = A[ij]$ ,
* where  $ij = i*n + j$ , which is the transpose of the normal
* column-wise storage.
*
* The contents of matrix A are destroyed.
*
* Set flag=0 to solve.
* Set flag=-1 to do a new back substitution for different B vector
* using the same A matrix previously reduced when flag=0.
*
* The routine returns nonzero on error; messages are printed.
*
*
* ACCURACY:
*
* Depends on the conditioning (range of eigenvalues) of matrix A.
*
*
* REFERENCE:
*
* Computer Solution of Linear Algebraic Systems,
* by George E. Forsythe and Cleve B. Moler; Prentice-Hall, 1967.

```

```

int simq( A, B, X, n, flag, IPS )
double A[], B[], X[];
int n, flag;
int IPS[];
{
    int i, j, ij, ip, ipj, ipk, ipn;
    int idxpiv, iback;
    int k, kp, kp1, kpj, kpj, kpj, kpj;
    int nip, nkp, nm1;
    double em, q, rownrm, big, size, pivot, sum;
    double abs();

```

```

    if( flag < 0 )
        goto solve;

```

```

/*      Initialize IPS and X      */

```

```

ij=0;
for( i=0; i<n; i++ )
{
    IPS[i] = i;
    rownrm = 0.0;
    for( j=0; j<n; j++ )
    {

```

(continued)

```

        q = abs( A[ij] );
        if( rownrm < q )
            rownrm = q;
        ++ij;
    }
    if( rownrm == 0.0 )
    {
        puts("SIMQ ROWNRM=0");
        return(1);
    }
    X[i] = 1.0/rownrm;
}

/*
/*      Gaussian elimination with partial pivoting      */
nm1 = n-1;
for( k=0; k<nm1; k++ )
{
    big = 0.0;
    for( i=k; i<n; i++ )
    {
        ip = IPS[i];
        ipk = n*ip + k;
        size = abs( A[ipk] ) * X[ip];
        if( size > big )
        {
            big = size;
            idxpiv = i;
        }
    }

    if( big == 0.0 )
    {
        puts( "SIMQ BIG=0" );
        return(2);
    }

    if( idxpiv != k )
    {
        j = IPS[k];
        IPS[k] = IPS[idxpiv];
        IPS[idxpiv] = j;
    }

    kp = IPS[k];
    kpk = n*kp + k;
    pivot = A[kpk];
    kp1 = k+1;
    for( i=kp1; i<n; i++ )
    {
        ip = IPS[i];
        ipk = n*ip + k;
        em = -A[ipk]/pivot;
        A[ipk] = -em;
        nip = n*ip;
        nkp = n*kp;
        for( j=kp1; j<n; j++ )
        {
            ipj = nip + j;
            A[ipj] = A[ipj] + em * A[nkp + j];
        }
    }

    kpn = n * IPS[n-1] + n - 1; /* last element of IPS[n] th row */
    if( A[kpn] == 0.0 )
    {
        puts( "SIMQ A[kpn]=0");
        return(3);
    }
}

/*
/*      back substitution      */

solve:
ip = IPS[0];
X[0] = B[ip];
for( i=1; i<n; i++ )
{

```



```

    ip = IPS[i];
    ipj = n * ip;
    sum = 0.0;
    for( j=0; j<i; j++ )
        {
            sum += A[ipj] * X[j];
            ++ipj;
        }
    X[i] = B[ip] - sum;
}

ipn = n * IPS[n-1] + n - 1;
X[n-1] = X[n-1]/A[ipn];

for( iback=1; iback<n; iback++ )
    {
        /* i goes (n-1),...,1 */
        i = nm1 - iback;
        ip = IPS[i];
        nip = n*ip;
        sum = 0.0;
        for( j=i+1; j<n; j++ )
            sum += A[nip+j] * X[j];
        X[i] = (X[i] - sum)/A[nip+i];
    }
return(0);
}

```

Listing4.pas

TEXT

Programming Project: "A Simple Windowing System, Part 2:
Implementation," Bruce Webster.
April, page 96. Listing 4, page 101. Apple Pascal.

```

    BBuf^[Offset+3] := Height;
    DoXfer(Save,Buffer);
    OldOff := Offset;
    Offset := Offset + Size;
    BBuf^[Offset-1] := OldOff div 256;
    BBuf^[Offset-2] := OldOff mod 256;
    BufUsed := True
end;
X1 := X1 * PPB; X2 := X2 * PPB;
Viewport(X1,X2,Y1,Y2);
FillScreen(White);
ViewPort(X1+2,X2-2,Y1+1,Y2-1);
FillScreen(Black)
end; { of proc OpenWindow }

procedure CloseWindow(var Error : Integer);
var
    X1,X2,Y1,Y2,Width,Height : Integer;
begin
    (* error checking again omitted for space *)
    Offset := OldOff;
    DoXfer(Restore,Buffer);
    If Offset = 0
        then BufUsed := False
        else with Buffer do
            OldOff := 256*BBuf^[Offset-1] + BBuf^[Offset-2];
        if BufUsed then with Buffer do begin
            X1 := BBuf^[OldOff] * PPB;
            Y1 := BBuf^[OldOff+1];
            Width := BBuf^[OldOff+2];
            Height:= BBuf^[OldOff+3];
            X2 := X1 + PPB*Width - 1;
            Y2 := Y1 + Height - 1;
            ViewPort(X1+2,X2-2,Y1+1,Y2-1)
        end
        else ViewPort(0,279,0,191)
    end; { of proc CloseWindow }

```

(continued)

```

procedure Initialize;
begin
  InitTurtle;
  Offset := 0; OldOff := 0;
  BufUsed := False
end; { of proc Initialize }

```

[LISTING 4]

```

const
  XMin    = 0;
  XMax    = 39;
  YMin    = 0;
  YMax    = 191;
  PPB     = 7;
  BufAddr = 16384;
  BufSize = 16383;

type
  Byte      = 0..255;
  Direction = (Save, Restore);
  ByteBuffer = packed array[0..BufSize] of Byte;
  BufRec     =
    record
      case Boolean of
        False : (Addr : Integer);
        True  : (BBuf : ^ByteBuffer)
      end;

var
  BufUsed      : Boolean;
  Buffer        : BufRec;
  Offset, OldOff : Integer;

function ScrAddr(Line : Integer) : Integer;
var
  Addr, Temp : Integer;
begin
  Line := 191 - Line;
  Addr := 8192 + 1024 * (Line mod 8);
  Temp := (Line div 8) mod 8;
  Line := Line div 64;
  ScrAddr := Addr + 128 * Temp + 40 * Line
end; { of func ScrAddr }

function DoXfer(Dir : Direction; var Buffer : BufRec);
var
  X1, Y1, Y2, Width, Height, BStart, Line : Integer;
  TBuf : BufRec;
begin
  with Buffer do begin
    X1 := BBuf^[Offset];
    Y1 := BBuf^[Offset+1];
    Width := BBuf^[Offset+2];
    Height := BBuf^[Offset+3];
    BStart := Offset + 4;
    Y2 := Y1 + Height - 1;
    for Line := Y1 to Y2 do begin
      TBuf.Addr := ScrAddr(Line);
      if Dir = Save
      then MoveLeft(TBuf.BBuf^[X1], BBuf^[BStart], Width)
      else MoveLeft(BBuf^[BStart], TBuf.BBuf^[X1], Width);
      BStart := BStart + Width
    end
  end
end; { of proc DoXfer }

procedure OpenWindow(X1, Y1, Width, Height : Integer;
                    var Error : Integer);
var
  Size, X2, Y2 : Integer;

```



```

begin
  Size := 6 + Width*Height;
  X2 := X1 + Width - 1;
  Y2 := Y1 + Height - 1;
  (* error checking goes here -- removed for space *)
  with Buffer do begin
    BBuf^[Offset] := X1;
    BBuf^[Offset+1] := Y1;
    BBuf^[Offset+2] := Width;
  end

```

chbev1.c

TEXT

"Computer Approximations," Stephen Moshier.
 April, page 161. Also download cheby.c, remes.c, and simq.c. For related
 modules, see the Cephes subsection of the C+UNIX file area.

```

0
*
*
*      Evaluate Chebyshev series
*
*
* SYNOPSIS:
*
*  int N;
*  double x, y, coef[N], chebev1();
*
*  y = chebev1( x, coef, N );
*
*
* DESCRIPTION:
*
*  Evaluates the series
*
*      N-1
*      -
*  y = > coef[i] Ti(x/2)
*      i=0
*
*  of Chebyshev polynomials Ti at argument x/2.
*
*  Coefficients are stored in reverse order, i.e. the zero
*  order term is last in the array. Note N is the number of
*  coefficients, not the order.
*
*  If coefficients are for the interval a to b, x must
*  have been transformed to x -> 2(2x - b - a)/(b-a) before
*  entering the routine. This maps x from (a, b) to (-1, 1),
*  over which the Chebyshev polynomials are defined.
*
*  If the coefficients are for the inverted interval, in
*  which (a, b) is mapped to (1/b, 1/a), the transformation
*  required is x -> 2(2ab/x - b - a)/(b-a). If b is infinity,
*  this becomes x -> 4a/x - 1.
*
*
* SPEED:
*
*  Taking advantage of the recurrence properties of the
*  Chebyshev polynomials, the routine requires one more
*  addition per. loop than evaluating a nested polynomial of
*  the same degree.
*
*/
/*

```

chbev1.c

*/

(continued)

```

/* Cephes Math Library Release 1.1:  March, 1985
 * Copyright 1985 by Stephen L. Moshier
 * Contributed to BIX for personal, noncommercial use only.
 * Direct inquiries to 30 Frost Street, Cambridge, MA 02140 */

```

```

double chbevl( x, array, n )
double x;
double array[];
int n;
{
double b0, b1, b2, *p;
int i;

p = array;
b0 = *p++;
b1 = 0.0;
i = n - 1;

do
    {
        b2 = b1;
        b1 = b0;
        b0 = x * b1 - b2 + *p++;
    }
while( --i );

return( 0.5*(b0-b2) );
}

```

cheby.c

TEXT
 "Computer Approximations," Stephen Moshier. See chbevl.c.

```

0      cheby.c
*
* Program to calculate coefficients of the Chebyshev polynomial
* expansion of a given input function. The algorithm computes
* the discrete Fourier cosine transform of the function evaluated
* at unevenly spaced points. Library routine chbevl.c uses the
* coefficients to calculate an approximate value of the original
* function.
* -- S. L. Moshier
*/

extern double PI;          /* 3.14159... */
extern double PI02;
double cosi[33] = {0.0,}; /* cosine array for Fourier transform */
double func[65] = {0.0,}; /* values of the function */

main()
{
double c, r, s, t, x, y, z, temp;
double low, high, dtemp;
long n;
int i, ii, j, n2, k, rr, invflg;
short *p;
char st[40];
double cos(), log(), exp(), sqrt();

low = 0.0;          /* low end of approximation interval */
high = 1.0;         /* high end */
invflg = 0;         /* set to 1 if inverted interval, else zero */
/* Note: inverted interval goes from 1/high to 1/low */
z = 0.0;
n = 64;             /* will find 64 coefficients */
/* but use only those greater than roundoff error */

n2 = n/2;
t = n;
t = PI/t;

```



```

/* calculate array of cosines */
puts("calculating cosines");
s = 1.0;
cosi[0] = 1.0;
i = 1;
while( i < 32 )
{
    y = cos( s * t );
    cosi[i] = y;
    s += 1.0;
    ++i;
}
cosi[32] = 0.0;

/*                                     cheby.c 2 */

/* calculate function at special values of the argument */
puts("calculating function values");
x = low;
y = high;
if( invflg && (low != 0.0) )
{
    /* inverted interval */
    temp = 1.0/x;
    x = 1.0/y;
    y = temp;
}
r = (x + y)/2.0;
printf( "center %.15E ", r);
s = (y - x)/2.0;
printf( "width %.15E\n", s);
i = 0;
while( i < 65 )
{
    if( i < n2 )
        c = cosi[i];
    else
        c = -cosi[64-i];
    temp = r + s * c;
/* if inverted interval, compute function(1/x) */
    if( invflg && (temp != 0.0) )
        temp = 1.0/temp;

    printf( "%.15E ", temp );

/* Insert call to function routine here: */
/*******/
    if( temp == 0.0 )
        y = 1.0;
    else
        y = exp( temp * log(2.0) );

/*******/
    func[i] = y;
    printf( "%.15E\n", y );
    ++i;
}

/*                                     cheby.c 3 */

puts( "calculating Chebyshev coefficients");
rr = 0;
while( rr < 65 )
{
    z = func[0]/2.0;
    j = 1;
    while( j < 65 )
    {
        k = (rr * j)/n2;
        i = rr * j - n2 * k;
        k &= 3;

        if( k == 0 )
            c = cosi[i];
        if( k == 1 )
            {

```

(continued)

```

        i = 32-i;
        c = -cosi[i];
        if( i == 32 )
            c = -c;
    }
    if( k == 2 )
    {
        c = -cosi[i];
    }
    if( k == 3 )
    {
        i = 32-i;
        c = cosi[i];
    }
    if( i != 32 )
    {
        temp = func[j];
        temp = c * temp;
        z += temp;
    }
    ++j;
}

if( i != 32 )
{
    temp /= 2.0;
    z = z - temp;
}

z *= 2.0;
temp = n;
z /= temp;
dtemp = z;
++rr;
sprintf( st, "/* %.16E */", dtemp );
puts( st );
}
}

```

atspgm.for

TEXT

"The ATOMCC Toolbox," Y. F. Chang. See atomcc.exe.

```

c*****
c   This program was produced by the  ATOMCC  translator version 7.10
c                                     Copyright (C) 1985, Y. F. Chang
c*****
c Portions (c) Copyright, Microsoft Corp., 1981. All rights reserved.
c This program was written for the following inputs
c
c FIRST PAINLEVE TRANSCENDENT
c   DIFF(Y,T,2) = 6.0*Y*Y + T
c-----
c no instructions in second input block
c-----
COMMON /IPASS/ LENSER,LENVAR,MPRINT,MSTIFF,LRUN,
+ KTRDCV,KNTSTP,KTSTIF,KXPNUM,KDIGS,KENDFG,NTERMS,NOPT
A /RPASS/ RADIUS,ERRLIM,ADJSTF,RCREAL,RCIMAG,RDCERR
B /CPASS/ START,END,ORDER
C /DPASS/ H,HNEW,XPRINT,DLTXPT
DIMENSION TMPS( 36, 1)
CHARACTER*6 NAMES
EQUIVALENCE (TMPS(1,1),Y(1))
DIMENSION NAMES(1), Y(36), T(2), TMPAAB(30), TMPAAA(30)
DATA NAMES(1)/'Y.....'/
Y(33) = 1.1
10 FORMAT(72H  ATOMCC  Ver. 7.10, Copyright (C) 1985, Y. F. Chang; S
   Aolution results./9H  *****)
11 FORMAT(/5X,11HStep number,I6,13H at the point,1P1E12.4/1X,
   A 9Hvalues of )
12 FORMAT(1X, A6,(1X,1P4E13. 5))
13 FORMAT(5X,21HStepsize adjusted to ,1PE13.5)

```



```

14 FORMAT(/5X,35HThe solution stopped normally after, I4,24H steps as
a set by nsteps. )
16 FORMAT(/5X,63HThe adjustment for stepsize seems to be in a loop. P
Alease try a /5X,22Hshorter series length. )
WRITE(*,10)
C-----
c Initialize variables to default values.
C-----
      NSTEPS = 40
      H = 1.E0
      ERR LIM = 1.E- 6
      LENSER = 30
      MPRINT = 4
      NTERMS = 2
      KTRDCV = 1
      ADJSTF = 1.E-2
      MSTIFF = 0
      DLTXP T = 0.E0
C-----
c start of third input block
C-----
C READ INTEGRATION INTERVAL AND INITIAL CONDITIONS.
      READ(5,1010) START,END,Y(1),Y(2)
1010 FORMAT(4F10.3)
      WRITE(*,1020) START,END,Y(1),Y(2)
1020 FORMAT(' SOLVE THE FIRST PAINLEVE TRANSCENDENT' /
+ ' INTERVAL: ',2F10.3 /
+ ' INITIAL CONDITIONS:',2F10.3 /)
C-----
c end of third input block
C-----
c More initializations
C-----
      DLTXP T = SIGN(DLTXP T,(END-START))
      H = SIGN(H,(END-START))
      KDIGS = 6
      XPRINT = START + DLTXP T
      KXPNUM = 35
      LENVAR = 36
      LRUN = 1
      KTSTIF = 0
      NUMEQS = 1
      IF(LENSER.GT.(LENVAR- 6)) LENSER = LENVAR - 6
      IF(MPRINT.LT.2) GO TO 17
      WRITE(*,11) KTSTIF,START
      K = Y(33)
      WRITE(*,12) NAMES(K),Y(1), Y(2)
C-----
c Loop for integration steps. Inside the loop, print the desired output
C-----
      17 DO 27 KINTS=1,NSTEPS
          KOUNT = 0
          KNTSTP = KINTS
      19 CONTINUE
          T(1) = START
          T(2) = H
          Y(2) = Y(2)*(H)
C-----
c Preliminary series calculations
C-----
          TMPAAA(1) = 6.E0*Y(1)
          TMPAAB(1) = TMPAAA(1)*Y(1)
          Y(3) = (TMPAAB(1) + T(1))*(H*H/2.E0)
          TMPAAA(2) = 6.E0*Y(2)
          TMPAAB(2) = TMPAAA(1)*Y(2) + TMPAAA(2)*Y(1)
          Y(4) = (TMPAAB(2) + T(2))*(H*H/6.E0)
C-----
c Loop for series calculations
C-----
          DO 23 K= 5,LENSER
              KA = K - 1
              KB = K - 2
              TMPAAA(KB) = 6.E0*Y(KB)
              TMPAAB(KB) = 0.E0
              KZ = 1 + KB
              DO 30 N=1, KB
                  L = KZ - N

```

```

      TMPAAB(KB) = TMPAAB(KB) + TMPAAA(N)*Y(L)
30      CONTINUE
      Y(K) = (TMPAAB(KB))*(H*H/(KB*KA))
c-----
c Test and adjust H to avoid over/under flow.
c-----
      IF(MSTIFF.GE.20 .AND. KTSTIF.GT.0) GO TO 23
      TMP = ABS(Y(K))
      IF(TMP.LE.1.E-35) GO TO 23
      IF(TMP.LT.1.E20 .AND. TMP.GT.1.E-20) GO TO 23
      IF(KTSTIF.NE.0 .AND. TMP.LT.1.0) GO TO 23
      KOUNT = KOUNT + 1
      IF(KOUNT.LT.9) GO TO 22
      WRITE(*,16)
      GO TO 28
22      CONTINUE
      Y(2) = Y(2)/(H)
      H = H * TMP**(0.3/(1-K))
      IF(MPRINT.GE.4) WRITE(*,13) H
      GO TO 19
23      LRUN = 1
c-----
c Calculate radius of convergence and take optimum step.
c-----
      CALL RDCV(TMPS,LENVAR,NUMEQS,NAMES)
24      CALL RSET(TMPS,LENVAR,NUMEQS,NAMES)
c-----
c no instructions in fourth input block
c-----
25      GO TO (26,28,24), KENDFG
26      H = SIGN(RADIUS,H)
      START = START + HNEW
      IF(MPRINT.LT.4) GO TO 27
      WRITE(*,11) KNTSTP, START
      K = Y(33)
      WRITE(*,12) NAMES(K),Y(1), Y(2)
27 CONTINUE
      WRITE(*,14) NSTEPS
28 CONTINUE
29 STOP
      END

```


manual.doc

BINARY

"The ATOMCC Toolbox," Y. F. Chang. See atomcc.exe.

ATOMCC USER MANUAL

For Micro-Computers on MSDOS

Version 7.10

by

Y. F. Chang

Copyright (C) 1983, 1985 Y. F. Chang

MSDOS is a registered trademark of Microsoft Corp.

Table of Contents

Chapter 1 Introduction	1
1.1 The Major Advancements in this Version	1
1.2 Purpose and Requirements of the Translator	1
1.3 Applicability	2
1.4 System Overview	4
1.4.1 The Translator, ATOMCC	4
1.4.2 The Object Program, ATSPGM	4
1.5 Purpose of the User Manual	5
1.6 Acknowledgements	6
1.7 References	6
Chapter 2 For New Users	8
2.1 Task of the ATOMCC Translator	8
2.2 Using the ATOMCC system	9
2.2.1 Step 1 - edit ODEINP	10
2.2.2 Step 2 - Run ATOMCC	14
2.2.3 Step 3 and 4 - Compile and link ATSPGM	15
2.2.4 Step 5 - Prepare the data	15
2.2.5 Step 6 - Run ATSPGM	16
2.3 Output at equally spaced points	17
2.3.1 ZEROT - Stopping at roots of variables	17
2.3.2 MSTIFF=20,21,22 - Stiff problems.	18
Chapter 3 How to Use ----	20
3.1 Solving your problem	20
3.1.1 Translator file, ODEINP	20
3.1.2 Translator file, the terminal	22
3.1.3 Translator file, ATSPGM	23
3.1.4 DATA input file	27
3.1.5 Solution file	28
3.1.6 User files	29
3.2 Using block 1	29
3.2.1 Format for the system of equations	30
3.2.2 Parameters in the equations	31
3.2.3 COPTION DOUBLE - Double-precision ATSPGM	31
3.2.4 COPTION COMPLX - Complex ATSPGM	32
3.2.5 COPTION DOUBLE, COMPLX - Double-complex ATSPGM	33
3.2.6 COPTION LENVAR=n - Series length	35
3.2.7 COPTION DUMP=n - Diagnostic messages	36

3.3 Using block 2	36
3.3.1 Subroutine form of ATSPGM	37
3.3.2 User declarations	37
3.3.3 Common blocks for user	38
3.4 Using block 3	38
3.4.1 Initial conditions	39
3.4.2 Parameters in the differential equations	39
3.4.3 Solve a problem repeatedly	40
3.4.4 START, END - Interval of integration	40
3.4.5 NSTEPS - Number of integration steps, default=40	40
3.4.6 H - Initial trial stepsize, default = 1.0	41
3.4.7 ERR LIM - Preset accuracy of the solution	41
3.4.8 ADJSTF - Error control for stiff problems	41
3.4.9 LENSER - Length of series used, default=30	42
3.4.10 MPRINT - Amount of print produced, default=4	42
3.4.11 DLTXT - Print point increments, default = 0.0	43
3.4.12 KTRDCV - Dynamic suppression of CALL RDCV	43
3.4.13 KPTS - Number of points on complex path	44
3.4.14 POINTS - Complex path of integration	44
3.4.15 MSTIFF=10 - Solutions which are entire	44
3.4.16 MSTIFF=20,21,22 - Stiff problems.	44
3.4.16.1 Steady-State Stiff Problems	45
3.5 Using block 4	46
3.5.1 Automatic printing of output points	46
3.5.2 User controlled printing of output points	47
3.5.3 Logarithmic spacing of output points	48
3.5.4 ZEROT - Stopping & printing at roots of variables	48
3.5.5 Finding singularities in real solutions	49
3.5.6 Stopping short of a singularity	51
3.6 Editing of ATSPGM	51
3.6.1 TERM - Fast generation of print at output points	51
3.7 Large systems	54
3.8 Solving ODE's in the complex domain	54

- 1 -

(Internal page reference for manual.doc)

Chapter 1

Introduction

This chapter is written to help you become familiar with the purpose and requirements of the ATOMCC system and with the organization of this manual.

1.1 The Major Advancements in this Version

The present version of ATOMCC, 7.10 for micros, contains a major advancement. Now, the ATOMCC system will solve stiff problems. This represents a significant departure from the central premise of the ATOMCC system, which is precise error control. For non-stiff problems, the user still have the most accurately controlled numerical method ever developed. For many problems, the precision is so good that there is ALMOST global error control.

For stiff problems, due to the nature of the "approximating" solution, there cannot be true error control. Therefore, the controlling parameter for errors in stiff problems is called ADJUSTF. It is only meant to be an adjusting constant that can be loosely referred to as an error control.

There is also another particularly useful feature in the current version. All the dependent variables are now placed into a temporary two-dimensional array (TMPS) by an EQUIVALENCE statement. This allows the user to reference each variable by an index value. For a system with x, y, and z as functions of t, the term y(5) can be also referred to as TMPS(5,2). Similarly, z(23) = TMPS(23,3).

1.2 Purpose and Requirements of the Translator

The ATOMCC system is a tool to be used in the solution of all initial value problems in ordinary differential equations, (stiff as well as non-stiff). It is simple enough to be used by students, practical enough to be used by engineers, and versatile enough to be used by research mathematicians.

The ATOMCC package is delivered on floppy disks. It uses Microsoft-FORTRAN77 (a registered trademark of Microsoft Corp.)

- 2 -

(Internal page reference for manual.doc)

and works on a micro-computer operating under MS-DOS. With the ATOMCC system, you now have in your possession a research tool whose capabilities far exceed those of standard numerical integration methods available on main-frame computers. To be able to run ATOMCC on your MSDOS micro-computer, you must have the following hardware and software:-

- an MSDOS computer, with an 8087 co-processor;
- at least 256K of RAM memory;
- two floppy disc drives, or a hard disc drive;
- the Microsoft-FORTRAN77 version 3.30.

A complete system includes the following disc files.

- The ATOMCC system files are:-

ATOMCC.EXE	This is the ATOMCC compiler that reads statements of differential equations and generates an object FORTRAN program called ATSPGM. The name ATSPGM for the object program file is fixed, but you may change it after it has been written by ATOMCC.
RDCV.OBJ	This is the ATOMCC subroutine library in single-precision.
DRDCV.OBJ	This is the ATOMCC subroutine library in double-precision.
CRDCV.OBJ	This is the ATOMCC subroutine library in complex.
CDRDCV.OBJ	This is the ATOMCC subroutine library in complex-double.

- The Microsoft-FORTRAN77 files are described in the Microsoft manual. The relevant files are:- FOR1.EXE, PAS2.EXE, LINK.EXE, and FORTRAN.LIB.

Throughout the discussions in this User Manual, we shall assume that all of the ATOMCC system files are on the A: floppy-disk drive, and the Microsoft-FORTRAN77 files are on the B: drive.

1.3 Applicability

- The ATOMCC method can solve:
 - * systems of stiff and non-stiff systems of initial value problems in ordinary differential equations in which

- 3 -

(Internal page reference for manual.doc)

- * the highest order derivative of each dependent variable is given explicitly on the left hand side of an equation, whose right hand side has a finite sequence of +, -, *, /, **, EXP, SIN, COS, TAN, SINH, COSH, TANH, ALOG, ACOS, ASIN, ATAN, or any function which is the solution to a differential equation.
- The known limitations of ATOMCC are:
 - * the derivatives may be of order at most 6, and
 - * there are at most 900 equations in the system.
- ATOMCC can also solve (with manual intervention):
 - * solutions which are polynomials,
 - * singular problems which require the application of l'Hopital's rule, or
 - * problems which have catastrophic subtractive errors in series generation.
- ATOMCC is most attractive for:-
 - * problems with stringent accuracy requirements,
 - * stiff problems,
 - * problems which must be solved repeatedly (such as parameter identification), or
 - * quick and easy problems (students' assignments).
- The very high order and precise error control used by ATOMCC have enabled it to solve many problems which other methods were unable to solve.
- The ATOMCC compiler allows for the solution of ODE's in the complex domain. This unique capability can be used to explore the structure of the singularities in the complex domain of non-linear problems. The analytic information about the location and order of singularities in the solution provides insight into the behavior of the system. This method has been used to map the first mathematical natural boundary discovered in the solution of a nonlinear dynamics problem (7).
- The complexity and execution time of ATSPGM depend on the number of functions and on the number of multiplications in the ODE system, not on the number of equations in the ODE system nor on the order of the derivatives involved. There is no penalty for high-order derivatives.
- As with all numerical methods, there is no substitute for insight into the structure of the ODE system and for the application of clever transformations.

- 4 -

(Internal page reference for manual.doc)

Solutions which are entire (have no singularities in the finite plane), should not be solved using the ATOMCC system. It is a total waste of computing power to solve linear problems using ATOMCC. This is particularly true for linear 'stiff' problems.

It can be EASILY show that ALL solutions that are entire can be solved in quasi-closed forms. This INCLUDES two-point boundary value problems!

Some special circumstances, which rarely occur, are identified either by ATOMCC or by the series analysis software, and an appropriate message is produced. In such cases, the user should examine the series (using MPRINT=10 in the third block), and seek the advice of the authors.

1.4 System Overview

1.4.1 The Translator, ATOMCC

The ATOMCC translator is an ODE-solving compiler written in FORTRAN. The ODE system to be solved is written into the ODEINP input file using conventions discussed in Chapters 2 and 3. The name ODEINP for the input file is fixed within ATOMCC; you must use this name. ATOMCC reads ODEINP and produces a FORTRAN object program, called ATSPGM. The name ATSPGM for the object program is also fixed; you must have a file by this name on your disc even if it is an empty file. The numerical solution to the ODE system is obtained by compiling and executing ATSPGM together with the library subroutine RDCV.OBJ or one of its variants.

ATOMCC accepts four blocks of data from ODEINP in which the user specifies the differential equations, the integration interval, initial conditions, and various other parameters to be used in the solution. The first block is used to specify the differential equations and commands to ATOMCC. The second through fourth blocks are used to insert statements directly into ATSPGM. The third block is required to specify the integration interval and the initial conditions. Detailed guidelines for the use of each block appear in Chapter 3.

1.4.2 The Object Program, ATSPGM

The ATSPGM object program implements the Taylor series algorithm for solving initial value problems in ordinary differential equations. This Taylor series algorithm is outlined below.

- Initialize method control parameters which may be modified.
- Assign initial conditions, the integration interval, and method control parameters.

- 5 -
(Internal page reference for manual.doc)

- Initialize method control parameters which may not be modified.
- Loop for each integration step.
 - * Initialize the first few series terms.
 - * Generate the entire series.
 - * Call subroutine RDCV to determine the optimal stepsize from (a) the location and order of the primary singularities, (b) the series length, (c) the error tolerance, and (d) adjust the stepsize.
 - * Call subroutine RSET to perform analytic continuation, and to print the solution.

In ATSPGM, the stepsize used to expand the series is related to the radius of convergence at each integration step. After a series is generated, the location and order of the primary singularity are calculated. Then, the stepsize is adjusted to control the error in the following manner. The terms of the series for a function $g(x)$ expanded at X_0 with a stepsize of $H := X - X_0$ are stored as reduced derivatives, $G(k+1) := G(k)(X_0) H^{**k}/k!$. The stepsize H can be varied to control the error by multiplying $G(k+1)$ by $(HNEW/H)^{**k}$.

An exception is made in the solution of stiff problems. The step-size is determined in stiff problems by the length of a polynomial that can adequately represent the function.

A method which uses an infinite Taylor series is A-stable; however, in practice the series must be truncated to N terms. Then, the characteristic polynomial is $p(x,y) = x - \text{Sum}[y(k)/k!]$. For example, the real-valued stability intervals are $(-8.85,0)$, $(-12.58,0)$, and $(-16.29,0)$ for $N = 20, 30$, and 40 , respectively. Taylor series methods are best suited to solve problems with high accuracy. However, since very high order derivatives are used in these methods, the solution of stiff problems can be easily solved using the approximation of a polynomial with an exponential.

1.5 Purpose of the User Manual

This ATOMCC User Manual is designed to support easy, and efficient use of the ATOMCC system. Chapter 2 may be used as a tutorial; the rest of this User Manual is written as a reference manual, not as a tutorial, so information is repeated as appropriate when it applies to different issues.

Chapter 1 presents an overview of the ATOMCC system to help you understand how its components fit together. This information is helpful to using the system as described in the rest of the manual. A more detailed discussion can be found in references (3) and (5).

- 6 -

(Internal page reference for manual.doc)

Chapter 2 is written as a tutorial for new users of the ATOMCC system. Its purpose is to show you how to use the ATOMCC system to solve initial value problems in ODE's. It assumes that you are familiar with FORTRAN programming, and with the concept of computing a solution to a system of ODE's. It gives examples showing how to solve some specific differential equations.

Chapter 3 is written for users who already have some experience using the ATOMCC system. This chapter is the heart of this User Manual. It attempts to show you how to use each of the features available from the ATOMCC translator and from the ATSPGM object program. It is organized for reference, not for sequential reading.

1.6 Acknowledgements

The author would like to express his gratitude to Roy Morris for the initial design and coding of the translator program, to students John Fauss, David Lowery, and Manuel Prieto for their work on series analysis, to Ray Moore, Mike Tabor, John Weiss, Mike Ziegler, Phil Bender, and others for many helpful suggestions, and to Jon Wright for the many suggestions which arose from his extensive use of the ATOMCC system. He is particularly indebted to Professor George Corliss, who assisted him in every aspect of this program.

1.7 References

Here we include some references which are relevant to the ATOMCC system. A more complete list of references appears in reference(5).

1. D. Barton, I. M. Willers, and R. V. M. Zahar, The automatic solution of ordinary differential equations by the method of Taylor series, Comput. J., v. 14, 1971, pp. 243-248.
2. Y. F. Chang, Automatic solution of differential equations, in Constructive and Computational Methods for Differential and Integral Equations, edited by D. L. Colton and R. P. Gilbert, Lecture Notes in Math., vol. 430, Springer-Verlag, New York, 1974, pp. 61-94.
3. Y. F. Chang and G. F. Corliss, Compiler for the solution of ordinary differential equations using Taylor series, Marquette University technical report, 1981.
4. Y. F. Chang and G. F. Corliss, Ratio-like and recurrence relation tests for convergence of series, J. Inst. Math. Appl., v. 25, 1980, pp. 349-359.

- 7 -

(Internal page reference for manual.doc)

5. Y. F. Chang and G. F. Corliss, Solving ordinary differential equations using Taylor series, ACM Trans. Math. Soft, v. 8, 1982, pp. 114-144.
6. Y. F. Chang, J. Fauss, M. Prieto and G. F. Corliss, Convergence analysis of compound Taylor series, Proceedings of the Seventh Conference on Numerical Mathematics and Computing, University of Manitoba, 1978, pp. 129-152.
7. Y. F. Chang, M. Tabor, J. Weiss, and G. Corliss, On the analytic structure of the Henon Heiles system, Phys. Lett. 85A (1981), pp. 211-213.
8. G. F. Corliss, Integrating ODE's in the complex plane - Pole vaulting, Math. Comp., v. 35, 1980, pp. 1181-1189.
9. G. F. Corliss and D. Lowery, Choosing a stepsize for Taylor series methods for solving ODE's, J. Comput. Appl. Math., v. 3, 1977, pp. 251-256.
10. R. E. Moore, Interval Analysis, Prentice-Hall, Englewood Cliffs, NJ, 1966.
11. L. B. Rall, Automatic Differentiation: Techniques and Applications, Lecture Notes in Computer Science #120, Springer-Verlag, Berlin, 1981.

- 8 -

(Internal page reference for manual.doc)

Chapter 2

For New Users

This Chapter is written for new users of the ATOMCC system. Its purpose is to show how to use this system to solve initial value problems in ordinary differential equations. Here, we give some specific examples of how to solve a system of differential equations. More detailed explanations are found in Chapter 3.

2.1 Task of the ATOMCC Translator

The ATOMCC system is a tool to help you solve differential equations. It consists of two major components:- a translator program (ATOMCC.EXE), and a subroutine object library (RDCV, DRDCV, CRDCV, CDRDCV). To understand the operation of these two components, you must first understand the six steps involved in using the system. We discuss the purpose of each step briefly, to acquaint you with the terms used in the detailed discussion in Section 2.2.

At Step 1 (edit ODEINP), the system of differential equations are stated in the form which ATOMCC expects. The input file ODEINP contains four separate blocks. (The name ODEINP is fixed within ATOMCC; so you must use this name.) The first block contains the differential and algebraic equations. ATOMCC compiler processes the data in this block to produce a FORTRAN object program ATSPGM which is then compiled and executed to solve the problem. (The name ATSPGM is also fixed within ATOMCC.) The second, third, and fourth blocks are copied unchanged from ODEINP to predetermined locations in ATSPGM.

At Step 2 (run ATOMCC), ATOMCC (a)reads the first block from ODEINP, (b)analyzes the differential equations, and (c)copies the second, third, and fourth blocks from ODEINP directly into ATSPGM at locations shown by examples below.

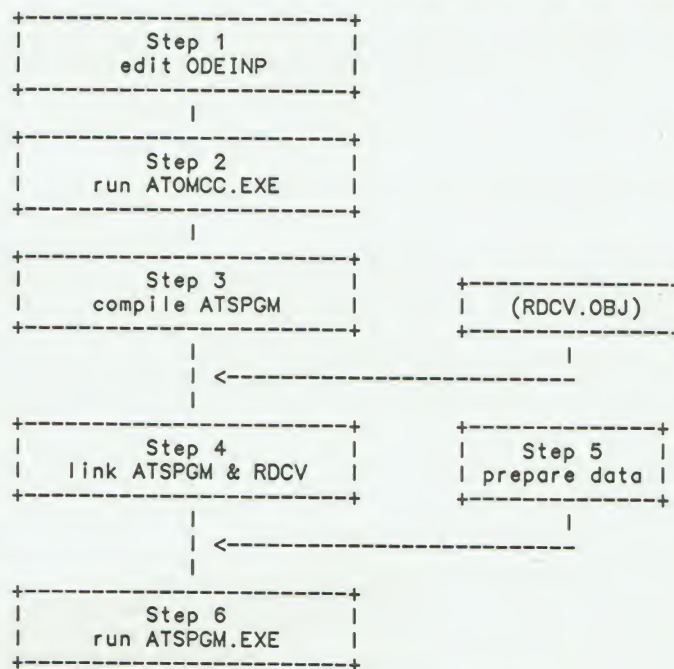
At Step 3 (compile ATSPGM), the ATSPGM program is compiled using Microsoft-FORTRAN ver 3.30 compiler.

At step 4 (link ATSPGM & RDCV), ATSPGM.OBJ is linked with the ATOMCC subroutine library RDCV.OBJ and FORTRAN.LIB to produce an executable module, ATSPGM.EXE.

The recommended manner to supply the initial conditions, the interval of integration, and control parameters is to read them

from a data file which you prepare at Step 5. The format of this data file is completely under your control, as shown by examples below. Step 5 may be done at any time before Step 6, and it may be omitted completely.

At Step 6 (run ATSPGM), the differential equations are solved. Each component of the equations is expanded in a Taylor series, and the solution point is moved forward by analytic continuation. ATSPGM reads the data file prepared at Step 5 and writes the solution results. The exact content, format, and location of the solution results depend on the data in ODEINP prepared at Step 1. Examples given below and in Chapter 3 show how this is done.



Steps for using the ATOMCC system

2.2 Using the ATOMCC system

In this section, we take you step-by-step through an example using the ATOMCC system.

2.2.1 Step 1 - edit ODEINP

The input file ODEINP specifies for ATOMCC

1. the system of differential equations to be solved,
2. how the initial conditions and the interval of integration are communicated to ATSPGM, and
3. the commands to control the operation of ATOMCC or to control the execution of ATSPGM.

The statements in ODEINP follows FORTRAN conventions. A "C" in column 1 denotes a COMMENT, columns 1-5 are used for label numbers, column 6 is used for continuation, columns 7-72 contain statements, and columns 73-80 are ignored. The statements in ODEINP may be in either upper-case or lower-case letters. In our discussions in this Manual, we use upper-case letters for emphasis. (One word of caution:- ATOMCC does not recognize tabs.)

Example 2-1. Simple ODEINP file.

```
C FIRST PAINLEVE TRANSCENDENT
  DIFF(Y,T,2) = 6.0*Y*Y + T      $
$
C READ INTEGRATION INTERVAL AND INITIAL CONDITIONS.
  READ(5,1010) START,END,Y(1),Y(2)
1010 FORMAT(4F10.3)
  WRITE(*,1020) START,END,Y(1),Y(2)
1020 FORMAT(' SOLVE THE FIRST PAINLEVE TRANSCENDENT' /
+ ' INTERVAL: ',2F10.3 /
+ ' INITIAL CONDITIONS:',2F10.3 /)      $
$
```

ODEINP must contain four blocks. Each block must terminate with the block terminator "\$" in columns 7-72. Blocks 2 and 4 are empty in Example 2-1 above.

The first block contains the system of differential equations. These equations are processed by ATOMCC to determine the recurrence relations that are written into ATSPGM to generate the Taylor series for each component of the solution. To enter the differential equations, DIFF(Y,X,N) is used to denote the N-th derivative of Y with respect to X. The value of N may range from 1 to 6, inclusively. The DIFF(,,) function is used to specify the system of ODE's with FORTRAN-like statements using standard FORTRAN operators and functions.

Rarely, ATOMCC may fail to produce the correct ATSPGM for your problem. In such a case, write your equations differently using many auxiliary variables. This will allow you to solve your problem; then, send a copy of the ODEINP that caused the problem to Y. F. Chang, Claremont McKenna College, Claremont, CA, 91711.

- 11 -

(Internal page reference for manual.doc)

The first block can also be used to control the operation of ATOMCC. The most commonly used option is for ATOMCC to write ATSPGM in double-precision with a "COPTION DOUBLE" card at the beginning of block 1.

Example 2-2. Double precision ATSPGM.

```
COPTION DOUBLE
C
C FIRST PAINLEVE TRANSCENDENT
      DIFF(Y,T,2) = 6.0*Y*Y + T      $
      $
C READ INTEGRATION INTERVAL AND INITIAL CONDITIONS.
      READ(5,1010) START,END,Y(1),Y(2)
1010  FORMAT(4F10.3)
      WRITE(*,1020) START,END,Y(1),Y(2)
1020  FORMAT(' SOLVE THE FIRST PAINLEVE TRANSCENDENT' /
+ ' INTERVAL: ',2F10.3 /
+ ' INITIAL CONDITIONS:',2F10.3 /)      $
      $
```

The second block is usually empty. It is used to insert non-executable FORTRAN statements at the beginning of ATSPGM, such as a SUBROUTINE card, a DIMENSION card, a COMMON card, etc.

The second, third, and fourth blocks are not processed syntactically by ATOMCC; they are copied directly from ODEINP into ATSPGM. Example 2-3 is the ATSPGM program written by ATOMCC for the ODEINP file shown in Example 2-1. Notice where block 3 is copied into an early part of ATSPGM.

Example 2-3. ATSPGM for Example 2-1.

```
c*****
c   This program was produced by the  ATOMCC  translator version 7.10
c                                     Copyright (C) 1985, Y. F. Chang
c*****
c Portions (c) Copyright, Microsoft Corp., 1981. All rights reserved.
c This program was written for the following inputs
c
C FIRST PAINLEVE TRANSCENDENT
C   DIFF(Y,T,2) = 6.0*Y*Y + T
c-----
c no instructions in second input block
c-----
      COMMON /IPASS/ LENSER,LENVAR,MPRINT,MSTIFF,LRUN,
+ KTRDCV,KNTSTP,KTSTIF,KXPNUM,KDIGS,KENDFG,NTERMS,NOPT
A /RPASS/ RADIUS,ERRLIM,ADJSTF,RCREAL,RCIMAG,RDCERR
B /CPASS/ START,END,ORDER
C /DPASS/ H,HNEW,XPRINT,DLTXPT
      DIMENSION TMPS( 36, 1)
      CHARACTER*6 NAMES
      EQUIVALENCE (TMPS(1,1),Y(1))
      DIMENSION NAMES(1), Y(36), T(2), TMPAAB(30), TMPAAA(30)
      DATA NAMES(1)/'Y.....'/
      Y(33) = 1.1
10  FORMAT(72H  ATOMCC  Ver. 7.10, Copyright (C) 1985, Y. F. Chang; S
```


- 12 -

(Internal page reference for manual.doc)

```

      Aolution results./9H  *****)
11 FORMAT(/5X,11HStep number,I6,13H at the point,1P1E12.4/1X,
   A 9Hvalues of )
12 FORMAT(1X, A6,(1X,1P4E13. 5))
13 FORMAT(5X,21HStepsize adjusted to ,1PE13.5)
14 FORMAT(/5X,35HThe solution stopped normally after, I4,24H steps as
   a set by nsteps. )
16 FORMAT(/5X,63HThe adjustment for stepsize seems to be in a loop. P
   Alease try a /5X,22Hshorter series length. )
      WRITE(*,10)
c-----
c Initialize variables to default values.
c-----
      NSTEPS = 40
      H = 1.E0
      ERR LIM = 1.E- 6
      LENSER = 30
      MPRINT = 4
      NTERMS = 2
      KTRDCV = 1
      ADJUSTF = 1.E-2
      MSTIFF = 0
      DLTXTPT = 0.E0
c-----
c start of third input block
c-----
C READ INTEGRATION INTERVAL AND INITIAL CONDITIONS.
      READ(5,1010) START,END,Y(1),Y(2)
1010 FORMAT(4F10.3)
      WRITE(*,1020) START,END,Y(1),Y(2)
1020 FORMAT(' SOLVE THE FIRST PAINLEVE TRANSCENDENT' /
+ ' INTERVAL: ',2F10.3 /
+ ' INITIAL CONDITIONS:',2F10.3 /)
c-----
c end of third input block
c-----
c More initializations
c-----
      DLTXTPT = SIGN(DLTXTPT,(END-START))
      H = SIGN(H,(END-START))
      KDIGS = 6
      XPRINT = START + DLTXTPT
      KXPNUM = 35
      LENVAR = 36
      LRUN = 1
      KTSTIF = 0
      NUMEQS = 1
      IF(LENSER.GT.(LENVAR- 6)) LENSER = LENVAR - 6
      IF(MPRINT.LT.2) GO TO 17
      WRITE(*,11) KTSTIF,START
      K = Y(33)
      WRITE(*,12) NAMES(K),Y(1), Y(2)
c-----
c Loop for integration steps. Inside the loop, print the desired output
c-----
17 DO 27 KINTS=1,NSTEPS
      KOUNT = 0
      KNTSTP = KINTS

```

- 13 -
(Internal page reference for manual.doc)

```

19  CONTINUE
    T(1) = START
    T(2) = H
    Y(2) = Y(2)*(H)
c-----
c Preliminary series calculations
c-----
    TMPAAA(1) = 6.E0*Y(1)
    TMPAAB(1) = TMPAAA(1)*Y(1)
    Y(3) = (TMPAAB(1) + T(1))*(H*H/2.E0)
    TMPAAA(2) = 6.E0*Y(2)
    TMPAAB(2) = TMPAAA(1)*Y(2) + TMPAAA(2)*Y(1)
    Y(4) = (TMPAAB(2) + T(2))*(H*H/6.E0)
c-----
c Loop for series calculations
c-----
    DO 23 K= 5,LENSER
      KA = K - 1
      KB = K - 2
      TMPAAA(KB) = 6.E0*Y(KB)
      TMPAAB(KB) = 0.E0
      KZ = 1 + KB
      DO 30 N=1, KB
        L = KZ - N
        TMPAAB(KB) = TMPAAB(KB) + TMPAAA(N)*Y(L)
      30  CONTINUE
      Y(K) = (TMPAAB(KB))*(H*H/(KB*KA))
c-----
c Test and adjust H to avoid over/under flow.
c-----
      IF(MSTIFF.GE.20 .AND. KTSTIF.GT.0) GO TO 23
      TMP = ABS(Y(K))
      IF(TMP.LE.1.E-35) GO TO 23
      IF(TMP.LT.1.E20 .AND. TMP.GT.1.E-20) GO TO 23
      IF(KTSTIF.NE.0 .AND. TMP.LT.1.0) GO TO 23
      KOUNT = KOUNT + 1
      IF(KOUNT.LT.9) GO TO 22
      WRITE(*,16)
      GO TO 28
    22  CONTINUE
      Y(2) = Y(2)/(H)
      H = H * TMP**(0.3/(1-K))
      IF(MPRINT.GE.4) WRITE(*,13) H
      GO TO 19
    23  LRUN = 1
c-----
c Calculate radius of convergence and take optimum step.
c-----
      CALL RDCV(TMPS,LENVAR,NUMEQS,NAMES)
    24  CALL RSET(TMPS,LENVAR,NUMEQS,NAMES)
c-----
c no instructions in fourth input block
c-----
    25  GO TO (26,28,24), KENDFG
    26  H = SIGN(RADIUS,H)
      START = START + HNEW
      IF(MPRINT.LT.4) GO TO 27
      WRITE(*,11) KNTSTP, START

```


- 14 -
(Internal page reference for manual.doc)

```

      K = Y(33)
      WRITE(*,12) NAMES(K),Y(1), Y(2)
27  CONTINUE
      WRITE(*,14) NSTEPS
28  CONTINUE
29  STOP
      END

```

The third block is usually used to specify the interval of integration and the initial conditions by reading them from a data file prepared at Step 5. This is the file DATA opened in block 3. The interval of integration is from START to END. END is allowed to be less than START for integration in a negative direction. The initial values (at START) of a dependent variable named y and its derivatives are assigned to the array Y as follows:-

```

Y(1) denotes y at START,
Y(2) denotes y' at START,
Y(3) denotes y'' at START, etc.

```

Thus in Example 2-1, two initial conditions Y(1) for $y(0)$ and Y(2) for $y'(0)$ are entered for the second order differential equation.

Any valid FORTRAN statement may be included in block 3 to be copied into ATSPGM, as shown in Example 2-1 by the WRITE statement to echo the input. The third block may also be used to change the default values of method-controlling variables. You can see in Example 2-3 that many variables are initialized immediately before block 3. The meaning of these variables is described in Chapter 3.

The fourth block is usually empty. It may be used to insert statements into ATSPGM at the end of each integration step.

This concludes the discussion of how to prepare the input file. More information about the use of specific features can be found in Chapter 3.

2.2.2 Step 2 - Run ATOMCC

The appropriate command to execute the ATOMCC compiler is simply [ATOMCC]. The ATOMCC translator uses two files:- ODEINP for the equation statements and initial conditions, and ATSPGM for the output object program. (The names ODEINP and ATSPGM are fixed within ATOMCC.) The messages produced by ATOMCC are placed on your terminal. To have the messages written onto a disc file (say MSG), use the command [ATOMCC > MSG].

- 15 -
(Internal page reference for manual.doc)

Example 2-4. Translator messages for Example 2-1.

ATOMCC Ver. 7.10, Copyright (C) 1985, Y. F. Chang.

Portions (c) Copyright, Microsoft Corp., 1981. All rights reserved.

```

FIRST PAINLEVE TRANSCENDENT
      DIFF(Y,T,2) = 6.0*Y*Y + T      $
equation 1 is in position 1
$
READ INTEGRATION INTERVAL AND INITIAL CONDITIONS.
      READ(5,1010) START,END,Y(1),Y(2)
1010 FORMAT(4F10.3)
      WRITE(*,1020) START,END,Y(1),Y(2)
1020 FORMAT(' SOLVE THE FIRST PAINLEVE TRANSCENDENT' /
+ ' INTERVAL:          ',2F10.3 /
+ ' INITIAL CONDITIONS:',2F10.3 /)      $
$
      ATOMCC completed
Stop - Program terminated.

```

2.2.3 Step 3 and 4 - Compile and link ATSPGM

As you get comfortable with the ATOMCC system, you will rarely inspect ATSPGM, unless either an error occurs, or you choose to edit ATSPGM by hand.

ATSPGM, written by ATOMCC, is just like any other FORTRAN program; you may edit it to suit your needs. Whether edited or not, ATSPGM is ready to be compiled and linked with the necessary subroutines from the ATOMCC library (RDCV).

The appropriate commands to compile and link ATSPGM are:-

- B:FOR1 ATSPGM. ;
- B:PAS2
- B:LINK ATSPGM+RDCV,,NUL,B:

2.2.4 Step 5 - Prepare the data

At Step 1, when you prepared ODEINP for ATOMCC, you may have included some READ statements in block 3 to communicate the interval of integration and the initial conditions to ATSPGM. Before you run ATSPGM, the data file to be read by those statements must be prepared with the appropriate file name given in your OPEN statement.

- 16 -
(Internal page reference for manual.doc)

Example 2-5. Data file for Example 2-1.

```
0.000    1.100    1.000    0.000
```

If you know that you will be solving a simple problem only once, Step 5 can be eliminated by stating the values of START, END, and the initial conditions with FORTRAN assignment statements in block 3 as shown below.

Example 2-6. Assignment statements in block 3.

```
C First Painleve transcendent
  DIFF(Y,T,2) = 6.0*Y*Y + T    $
$
C Assign integration interval and initial conditions.
  START = 0.0
  END = 1.1
  Y(1) = 1.0
  Y(2) = 0.0
  WRITE(*,1020) START,END,Y(1),Y(2)
1020 FORMAT(' Solve the first Painleve transcendent' /
+ ' Interval: ',2F10.3 /
+ ' Initial conditions:',2F10.3 /)    $
$
```

2.2.5 Step 6 - Run ATSPGM

At step 6, you are ready to run ATSPGM; the command is simply [ATSPGM]. ATSPGM writes its output to your terminal, for solution output on a disc file (say PRTOUT) use [ATSPGM > PRTOUT].

Example 2-7. Solution Output for Example 2-1.

```
ATOMCC Ver. 7.10, Copyright (C) 1985, Y. F. Chang; Solution results.
*****
SOLVE THE FIRST PAINLEVE TRANSCENDENT
INTERVAL:          .000    1.100
INITIAL CONDITIONS: 1.000    .000

  Step number      0 at the point .0000E+00
values of
Y..... 1.00000E+00 .00000E+00

  Step number      1 at the point 7.1000E-01
values of
Y..... 4.04877E+00 1.62701E+01
```

- 17 -
(Internal page reference for manual.doc)

```

      Step number      2 at the point  1.0100E+00
values of
Y.....  2.58324E+01  2.62656E+02

```

```

      Step number      3 at the point  1.1000E+00
values of
Y.....  8.77693E+01  1.64459E+03
Stop - Program terminated.

```

2.3 Output at equally spaced points

You should be able to use ATOMCC to solve routine problems. The points at which ATSPGM computes the solutions are determined by the actual integration steps taken, which are not uniform in size. This Section shows you how to force ATSPGM to print the solutions at equally spaced points.

The output from ATSPGM is controlled by two variables, MPRINT (amount of print), and DLTXT (print interval). To produce output at equally spaced points, assign MPRINT=2 to turn off the print at the actual integration steps, and assign DLTXT=DELTA, where DELTA is your desired print interval.

Example 2-8. Equally spaced output points.

```

c First Painleve transcendent
  DIFF(Y,T,2) = 6.0*Y*Y + T      $
$
c Assign integration interval and initial conditions.
  MPRINT = 2
  DLTXT = 0.2
  START = 0.0
  END = 1.1
  Y(1) = 1.0
  Y(2) = 0.0
  WRITE(*,1020) START,END,Y(1),Y(2)
1020 FORMAT(' Solve the first Painleve transcendent' /
+ ' Interval:           ',2F10.3 /
+ ' Initial conditions:',2F10.3 /)      $
$

```

2.3.1 ZEROT - Stopping at roots of variables

It is often of interest to locate points at which a component of the solution has a root or assumes some specified value. The subroutine ZEROT automatically solve such problems. DZEROT is the double-precision version.

- 18 -
(Internal page reference for manual.doc)

The form of the CALL is

```
CALL ZEROT(NUMBER,Y,ROOT,KEY,TMPS,LENVAR,NUMEQS)
```

where

NUMBER is the index of the Y series term whose root is sought,
Y is the variable whose root is desired,
ROOT is the value Y is to assume (= 0 for a root),
KEY is 1 if Y is a dependent variable, or
0 if Y is not a dependent variable.

The arguments TMPS, LENVAR, and NUMEQS must be exactly as written above.

Example 3-16. Rootfinding with ZEROT.

```
DIFF(Y,T,2) = 6*Y*Y + T $
$
START = 0.0
END = 1.15
ROOT = 20.0
Y(1) = 1.0
Y(2) = 0.0
$
CALL ZEROT(1,Y,ROOT,1)
$
```

When the variable whose root is being sought is not a dependent variable, KEY is set to 0.

```
CALL ZEROT(2,VARY,0.0,0)
IF(LRUN.NE.0) GO TO 25
TEMP = START + HNEW
WRITE(7,1010) KINTS,TEMP,VARY(1),VARY(2)
1010 FORMAT(I5,3F10.4)
GO TO 25 $
```

In these examples, it is not necessary for one to print the information as shown in the second case. The ATSPGM program does stop and restart the solution automatically at the exact root and the output is controlled by MPRINT.

The index NUMBER can be set at any positive (non-zero) integer value; however, obviously when NUMBER is very large the accuracy of the root will suffer.

2.3.2 MSTIFF=20,21,22 - Stiff problems.

This version of ATOMCC contains a double-precision algorithm to solve stiff problems. To use it, one can either set MSTIFF=20, or 21, or 22. Other parameters that should be controlled are H, ADJSTF, and NSTEPS. It is also desirable to set MPRINT to 7, at least initially, for observing the progress of the solution. If it should be evident that the problem is not really stiff, then it is most advisable to solve it as a normal problem.

- 19 -
(Internal page reference for manual.doc)

MSTIFF=20 is the more conservative of the three algorithms. In this case, LENSER is set to be 15. The default value for ADJSTF, the error-controlling parameter, is a rather large 1.E-2. The user should run the stiff solution at least one more time with a somewhat smaller ADJSTF, say 1.E-3, to check on its validity.

When MSTIFF=21, LENSER is set to only 10. So, this option should be used only if the user is absolutely certain that the problem under study is very stiff. The solution of stiff problems under this option is considerably faster than that for MSTIFF=20.

MSTIFF=22 is identical to MSTIFF=20 except for the fact that there is no attempt to identify steady-state solutions.

As mentioned above, the stiff algorithm is written in double-precision. It is simply not cost effective to solve such problems using single-precision. There is one other restriction on stiff problems. All such problems must be stated as first-degree ODE's.

- 20 -
(Internal page reference for manual.doc)

Chapter 3

How to Use ----

This Chapter is written for users who already have solved several problems using the ATOMCC system. It assumes familiarity with FORTRAN programming, with Chapter 2, and with the numerical solution of ODE's. This Chapter is the heart of this User Manual. It attempts to show how to use each of the features available from ATOMCC and from ATSPGM. It is organized for reference, not for sequential reading. Consequently, some information found in other parts of this Manual are repeated here.

3.1 Solving your problem

The tasks which must be accomplished in order to run the ATOMCC system on your computer were discussed in Section 2.2. They are:-

Edit	ODEINP	(containing ODE's)
Run	ATOMCC.EXE	(execute ATOMCC translator) (This creates ATSPGM, the object FORTRAN program, which is treated like any FORTRAN program. ATSPGM may be edited.)
Compile	ATSPGM.	(This creates ATSPGM.OBJ, the object module.)
Link	ATSPGM.OBJ,	with library options (RDCV.OBJ for single precision DRDCV.OBJ for double precision CRDCV.OBJ for complex, and CDRDCV.OBJ for complex double) (This creates ATSPGM.EXE, the execution module.)
Edit	DATA	input-file if any is used.
Run	ATSPGM.EXE	

3.1.1 Translator file, ODEINP

The ODEINP file contains the ODE's to be solved and information specifying how the initial conditions and the integration interval are determined in ATSPGM. It may contain commands to control (a) the operation of the ATOMCC translator, (b) the execution of the

- 21 -
(Internal page reference for manual.doc)

solution, and (c) the desired format of the output. The names ODEINP and ATSPGM are fixed within ATOMCC; you must have these files on your disc when you execute ATOMCC.

The data in ODEINP follows FORTRAN conventions. Columns 1-5 are used for line numbers, column 6 is used for continuation characters, columns 7-72 contain statements, and columns 73-80 are ignored. As in FORTRAN, all blanks are ignored. A 'C' in column 1 denotes a comment which is copied directly into the ATSPGM file. The statements in ODEINP can be either upper-case or lower-case letters. We use upper-case in this manual for emphasis. (A word of caution, the tab character is not recognized by ATOMCC.)

The ODEINP file contains four blocks. Each block ends with the block terminator symbol '\$' in columns 73-80. (A comment card must not contain a block terminator '\$'.) Sections 3.2-3.4 discuss each of the blocks in detail. Here is an example of an input file which illustrates several of the features which will be discussed in Sections 3.2-3.5.

Example 3-1. ODEINP file.

```
C Block 1
C
C System with parameter.
C
      DIFF(X,T,2) = - ALPHA*X*R
      DIFF(Y,T,2) = - ALPHA*Y*R
      R = (X*X + Y*Y)**(-1.5)
      ALPHA = 0.65          $

c
c Block 2
c
      CHARACTER*80 LINE      $

c
c Block 3
c
c Read:- heading line, print code, maximum number of integration
c       steps.
c Echo the above.
c Read:- heading line, integration interval, print interval,
c       parameter in equations, initial conditions.
c Echo the above.
c
      OPEN(5,FILE='DATA')
      OPEN(7,FILE='PLOTS',STATUS='NEW')
      READ(5,1010) LINE,MPRINT,NSTEPS
1010 FORMAT(A80/2I10)
      WRITE(*,1010) LINE,MPRINT,NSTEPS
      READ(5,1020) LINE,START,END,DLTXPT,ALPHA,X(1),X(2),Y(1),Y(2)
1020 FORMAT(A80/8F10.3)
      WRITE(*,1020) LINE,START,END,DLTXPT,ALPHA,X(1),X(2),Y(1),Y(2)
c Assignment statements for the error control parameter
      ERR LIM = 1.0E-04      $
```


- 22 -
(Internal page reference for manual.doc)

```
c
c Block 4
c
c Produce file of data for plotting.
c
      IF(KENDFG.EQ.3) WRITE(7,1030) KINTS,XPRINT,X(1),X(2),Y(1),Y(2)
1030 FORMAT(I5,1P5E14.5)      $
```

If you have a simple problem which will be solved only once, the contents of the 'DATA' file may be entered directly as data into block 3. However, the compilation and linking of the ATSPGM file takes an appreciable amount of time and therefore should be avoided on problems that is solved more than once.

There are two ways to solve a given problem containing functions. Either they can be placed as FORTRAN statements in ODEINP to be copied directly into ATSPGM, or they can be inserted by editing ATSPGM. The choice depends on your personal style; the authors prefer to work with ODEINP, because it is short. Therefore, it is easy to find the correct place to make changes. The cost of re-running the ATOMCC translator is well worth the convenience, because ATOMCC is very fast.

3.1.2 Translator file, the terminal

The translator messages contains the information which the ATOMCC expects you to inspect. It includes an echo of the input file and any error messages. The Appendix contains a list of the error messages which may be produced. The messages will appear on your terminal.

Example 3-2. Translator messages for Example 3-1.

ATOMCC Ver. 7.10, Copyright (C) 1985, Y. F. Chang.

Portions (c) Copyright, Microsoft Corp., 1981. All rights reserved.

```
c Block 1
c System with parameter.
      DIFF(X,T,2) = - ALPHA*X*R
      DIFF(Y,T,2) = - ALPHA*Y*R
      R = (X*X + Y*Y)**(-1.5)
      ALPHA = 0.65      $
equation 3 is in position 1
equation 4 is in position 2
equation 1 is in position 3
equation 2 is in position 4
```

```
c Block 2
      CHARACTER*80 LINE      $
```

- 23 -
(Internal page reference for manual.doc)

c Block 3

```
c Read:- heading line, print code, maximum number of integration
c steps.
c Echo the above.
c Read:- heading line, integration interval, print interval,
c parameter in equations, initial conditions.
c Echo the above.
```

```
OPEN(5,FILE='DATA')
OPEN(7,FILE='PLOTS',STATUS='NEW')
READ(5,1010) LINE,MPRINT,NSTEPS
1010 FORMAT(A80/2I10)
WRITE(*,1010) LINE,MPRINT,NSTEPS
READ(5,1020) LINE,START,END,DLTXPT,ALPHA,X(1),X(2),Y(1),Y(2)
1020 FORMAT(A80/8F10.3)
WRITE(*,1020) LINE,START,END,DLTXPT,ALPHA,X(1),X(2),Y(1),Y(2)
c Assignment statements for the error control parameter
ERRLIM = 1.0E-04 $
```

c Block 4

c Produce file of data for plotting.

```
IF(KENDFG.EQ.3) WRITE(7,1030)KINTS,XPRINT,X(31),X(32),Y(31),Y(32)
1030 FORMAT(I5,1P5E14.5) $
ATOMCC completed
Stop - Program terminated.
```

3.1.3 Translator file, ATSPGM

The ATSPGM file contains the FORTRAN object program written by ATOMCC to solve the system of differential equations using long Taylor series. ATOMCC uses the variable names given in ODEINP, so that ATSPGM appears to have been custom written for the specific problem. Usually you do not need to inspect ATSPGM, but sometimes you may find it necessary to edit it like you would edit any other FORTRAN program to achieve some particular result. Section 3.6 contains an example for which editing of ATSPGM is necessary.

Example 3-3. ATSPGM for Example 3-1.

```
c*****
c This program was produced by the ATOMCC translator version 7.10
c Copyright (C) 1985, Y. F. Chang
c*****
c Portions (c) Copyright, Microsoft Corp., 1981. All rights reserved.
c This program was written for the following inputs
c
c BLOCK 1
c SYSTEM WITH PARAMETER.
c DIFF(X,T,2) = - ALPHA*X*R
c DIFF(Y,T,2) = - ALPHA*Y*R
c R = (X*X + Y*Y)**(-1.5)
c ALPHA = 0.65
c-----
```


- 24 -
(Internal page reference for manual.doc)

```

c start of second input block
c-----
c
c Block 2
c
      CHARACTER*80 LINE
c-----
c end of second input block
c-----
      COMMON /IPASS/ LENSER,LENVAR,MPRINT,MSTIFF,LRUN,
+ KTRDCV,KNTSTP,KTSTIF,KXPNUM,KDIGS,KENDFG,NTERMS,NOPT
      A /RPASS/ RADIUS,ERRLIM,ADJSTF,RCREAL,RCIMAG,RDCERR
      B /CPASS/ START,END,ORDER
      C /DPASS/ H,HNEW,XPRINT,DLTXPT
      DIMENSION TMPS( 36, 2)
      CHARACTER*6 NAMES
      EQUIVALENCE (TMPS(1,1),X(1)),(TMPS(1,2),Y(1))
      DIMENSION NAMES(2), T(2), Y(36), X(36), R(30), TMPAAH(30),
      A TMPAAG(30), TMPAAF(30), TMPAAE(30), TMPAAD(30), TMPAAC(30),
      B TMPAAB(30)
      DATA NAMES(1)/'X.....'/
      DATA NAMES(2)/'Y.....'/
      X(33) = 1.1
      Y(33) = 2.1
10  FORMAT(72H  ATOMCC  Ver. 7.10, Copyright (C) 1985, Y. F. Chang; S
      Aolution results./9H  *****)
11  FORMAT(/5X,11HStep number,I6,13H at the point,1P1E12.4/1X,
      A 9Hvalues of )
12  FORMAT(1X, A6,(1X,1P4E13. 5))
13  FORMAT(5X,21HStepsize adjusted to ,1PE13.5)
14  FORMAT(/5X,35HThe solution stopped normally after, I4,24H steps as
      a set by nsteps. )
16  FORMAT(/5X,63HThe adjustment for stepsize seems to be in a loop. P
      Alease try a /5X,22Hshorter series length. )
      WRITE(*,10)
c-----
c Initialize variables to default values.
c-----
      NSTEPS = 40
      H = 1.E0
      ERRLIM = 1.E- 6
      LENSER = 30
      MPRINT = 4
      NTERMS = 2
      KTRDCV = 2
      ADJSTF = 1.E-2
      MSTIFF = 0
      DLTXT = 0.E0
c-----
c constant expressions
c-----
      ALPHA = 6.5E-1
c-----
c start of third input block
c-----
c
c Block 3
c

```

- 25 -
(Internal page reference for manual.doc)

```

c Read:- heading line, print code, maximum number of integration
c steps.
c Echo the above.
c Read:- heading line, integration interval, print interval,
c parameter in equations, initial conditions.
c Echo the above.
c
      OPEN(5,FILE='DATA')
      OPEN(7,FILE='PLOTS',STATUS='NEW')
      READ(5,1010) LINE,MPRINT,NSTEPS
1010 FORMAT(A80/2I10)
      WRITE(*,1010) LINE,MPRINT,NSTEPS
      READ(5,1020) LINE,START,END,DLTXPT,ALPHA,X(1),X(2),Y(1),Y(2)
1020 FORMAT(A80/8F10.3)
      WRITE(*,1020) LINE,START,END,DLTXPT,ALPHA,X(1),X(2),Y(1),Y(2)
c Assignment statements for the error control parameter
      ERR LIM = 1.0E-04
c-----
c end of third input block
c-----
      TMPAAA = -1.5E0
c-----
c constant expressions
c-----
c More initializations
c-----
      DLTXPT = SIGN(DLTXPT,(END-START))
      H = SIGN(H,(END-START))
      KDIGS = 6
      XPRINT = START + DLTXPT
      KXPNUM = 35
      LENVAR = 36
      LRUN = 1
      KTSTIF = 0
      NUMEQS = 2
      IF(LENSER.GT.(LENVAR- 6)) LENSER = LENVAR - 6
      IF(MPRINT.LT.2) GO TO 17
      WRITE(*,11) KTSTIF,START
      K = X(33)
      WRITE(*,12) NAMES(K),X(1), X(2)
      K = Y(33)
      WRITE(*,12) NAMES(K),Y(1), Y(2)
c-----
c Loop for integration steps. Inside the loop, print the desired output
c-----
      17 DO 27 KINTS=1,NSTEPS
          KOUNT = 0
          KNTSTP = KINTS
      19 CONTINUE
          T(1) = START
          T(2) = H
          X(2) = X(2)*(H)
          Y(2) = Y(2)*(H)
c-----
c Preliminary series calculations
c-----
          TMPAAB(1) = X(1)*X(1)
          TMPAAC(1) = Y(1)*Y(1)

```


- 26 -
(Internal page reference for manual.doc)

```

TMPAAE(1) = ALPHA*X(1)
TMPAAG(1) = ALPHA*Y(1)
TMPAAD(1) = TMPAAB(1) + TMPAAC(1)
R(1) = TMPAAD(1) ** TMPAAA
TMPAAF(1) = TMPAAE(1)*R(1)
TMPAAH(1) = TMPAAG(1)*R(1)
X(3) = (-TMPAAF(1))*(H*H/2.E0)
Y(3) = (-TMPAAH(1))*(H*H/2.E0)
TMPAAB(2) = X(1)*X(2) + X(2)*X(1)
TMPAAC(2) = Y(1)*Y(2) + Y(2)*Y(1)
TMPAAE(2) = ALPHA*X(2)
TMPAAG(2) = ALPHA*Y(2)
TMPAAD(2) = TMPAAB(2) + TMPAAC(2)
R(2) = TMPAAA*R(1)*TMPAAD(2)/TMPAAD(1)
TMPAAF(2) = TMPAAE(1)*R(2) + TMPAAE(2)*R(1)
TMPAAH(2) = TMPAAG(1)*R(2) + TMPAAG(2)*R(1)
X(4) = (-TMPAAF(2))*(H*H/6.E0)
Y(4) = (-TMPAAH(2))*(H*H/6.E0)
c-----
c Loop for series calculations
c-----
DO 23 K= 5,LENSER
  KA = K - 1
  KB = K - 2
  KC = K - 3
  TMPAAB(KB) = 0.E0
  TMPAAC(KB) = 0.E0
  KZ = 1 + KB
  DO 30 N=1, KB
    L = KZ - N
    TMPAAB(KB) = TMPAAB(KB) + X(N)*X(L)
    TMPAAC(KB) = TMPAAC(KB) + Y(N)*Y(L)
  30    CONTINUE
    TMPAAE(KB) = ALPHA*X(KB)
    TMPAAG(KB) = ALPHA*Y(KB)
    TMPAAD(KB) = TMPAAB(KB) + TMPAAC(KB)
    R(KB) = R(1)*TMPAAD(KC+1)*(KC)*TMPAAA
    KY = 2 + KC
    DO 31 N=2, KC
      L = KY - N
      AL = (L - 1)
  31    R(KB) = R(KB) + R(N)*TMPAAD(L)*AL
      A *TMPAAA - TMPAAD(N)*R(L)*AL
      R(KB) = R(KB)/(KC)/TMPAAD(1)
      TMPAAF(KB) = 0.E0
      TMPAAH(KB) = 0.E0
      KZ = 1 + KB
      DO 32 N=1, KB
        L = KZ - N
        TMPAAF(KB) = TMPAAF(KB) + TMPAAE(N)*R(L)
        TMPAAH(KB) = TMPAAH(KB) + TMPAAG(N)*R(L)
      32    CONTINUE
      X(K) = (-TMPAAF(KB))*(H*H/(KB*KA))
      Y(K) = (-TMPAAH(KB))*(H*H/(KB*KA))
c-----
c Test and adjust H to avoid over/under flow.
c-----
IF(MSTIFF.GE.20 .AND. KTSTIF.GT.0) GO TO 23

```

- 27 -
(Internal page reference for manual.doc)

```

      TMP = ABS(X(K))
      IF(TMP.LE.1.E-35) GO TO 23
      IF(TMP.LT.1.E20 .AND. TMP.GT.1.E-20) GO TO 23
      IF(KTSTIF.NE.0 .AND. TMP.LT.1.0) GO TO 23
      KOUNT = KOUNT + 1
      IF(KOUNT.LT.9) GO TO 22
      WRITE(*,16)
      GO TO 28
22    CONTINUE
      X(2) = X(2)/(H)
      Y(2) = Y(2)/(H)
      H = H * TMP**(0.3/(1-K))
      IF(MPRINT.GE.4) WRITE(*,13) H
      GO TO 19
23    LRUN = 1
c-----
c Calculate radius of convergence and take optimum step.
c-----
      CALL RDCV(TMPS,LENVAR,NUMEQS,NAMES)
24    CALL RSET(TMPS,LENVAR,NUMEQS,NAMES)
c-----
c start of fourth input block
c-----
c
c Block 4
c
c Produce file of data for plotting.
c
      IF(KENDFG.EQ.3) WRITE(7,1030)KINTS,XPRINT,X(31),X(32),Y(31),Y(32)
1030 FORMAT(I5,1P5E14.5)
c-----
c end of fourth input block
c-----
25    GO TO (26,28,24), KENDFG
26    H = SIGN(RADIUS,H)
      START = START + HNEW
      IF(MPRINT.LT.4) GO TO 27
      WRITE(*,11) KNTSTP, START
      K = X(33)
      WRITE(*,12) NAMES(K),X(1), X(2)
      K = Y(33)
      WRITE(*,12) NAMES(K),Y(1), Y(2)
27    CONTINUE
      WRITE(*,14) NSTEPS
28    CONTINUE
29    STOP
      END

```

3.1.4 DATA input file

For most problems solved using the ATOMCC system, you should communicate the initial conditions, the integration interval, the coefficients in the differential equations, and the ATOMCC control parameters by reading them from a DATA input file.

- 28 -
(Internal page reference for manual.doc)

Example 3-4. DATA input file for Example 3-1.

```

MPRINT  NSTEPS          for example 3-1
  4      80
START    END    DLTXPT  ALPHA    X(1)    X(2)    Y(1)    Y(2)
1.0      10.0    0.25    0.58    -1.0     0.0     0.0     4.3

```

3.1.5 Solution file

ATSPGM writes all of its messages and answers to your terminal. The format of the solution depends on the ATOMCC control parameters (see Section 3.4). If you should wish to have the solution written onto a disc file, you can use the execution statement (ATSPGM > PRTOUT). Then, all messages and answers will be written to the solution file PRTOUT. The solution file can also contain information written by user supplied WRITE(*,xxx) statements. A portion of the solution file is given below.

Example 3-5. Solution file for Example 3-1.

ATOMCC Ver. 7.10, Copyright (C) 1985, Y. F. Chang; Solution results.

```

*****
MPRINT  NSTEPS          for example 3-1
  4      80
START    END    DLTXPT  ALPHA    X(1)    X(2)    Y(1)    Y(2)
1.000    10.000   .250    .580   -1.000   .000   .000   4.300

```

```

Step number      0 at the point  1.0000E+00
values of
X..... -1.00000E+00   .00000E+00
Y.....  .00000E+00   4.30000E+00

```

```

Step number      1 at the point  1.1430E+00
values of
X..... -9.94536E-01   7.08452E-02
Y.....  6.13850E-01   4.27990E+00

```

```

Step number      2 at the point  1.2500E+00
values of
X..... -9.85268E-01   9.92472E-02
Y.....  1.07052E+00   4.25646E+00

```

```

Step number      2 at the point  1.3400E+00
values of
X..... -9.75709E-01   1.11976E-01
Y.....  1.45285E+00   4.24032E+00

```

```

Step number      3 at the point  1.5000E+00
values of
X..... -9.56779E-01   1.23037E-01
Y.....  2.12958E+00   4.22039E+00

```

- 29 -
(Internal page reference for manual.doc)

Step number 3 at the point 1.6400E+00
values of
X..... -9.39208E-01 1.27496E-01
Y..... 2.71959E+00 4.20915E+00

Step number 4 at the point 1.7500E+00
values of
X..... -9.25064E-01 1.29523E-01
Y..... 3.18223E+00 4.20277E+00

Step number 4 at the point 2.0000E+00
values of
X..... -8.92333E-01 1.31982E-01
Y..... 4.23159E+00 4.19295E+00

Step number 4 at the point 2.1300E+00
values of
X..... -8.75128E-01 1.32677E-01
Y..... 4.77644E+00 4.18942E+00

3.1.6 User files

It is often useful to create your own output files using WRITE statements to produce output in a format of your choice. Example 3-6 shows a portion of such a file for plotting, produced by Example 3-1.

Example 3-6. User file for plotting.

2	1.25000E+00	-9.85268E-01	9.92472E-02	1.07052E+00	4.25646E+00
3	1.50000E+00	-9.56779E-01	1.23037E-01	2.12958E+00	4.22039E+00
4	1.75000E+00	-9.25064E-01	1.29523E-01	3.18223E+00	4.20277E+00
4	2.00000E+00	-8.92333E-01	1.31982E-01	4.23159E+00	4.19295E+00
5	2.25000E+00	-8.59178E-01	1.33133E-01	5.27900E+00	4.18678E+00
5	2.50000E+00	-8.25810E-01	1.33750E-01	6.32514E+00	4.18258E+00
6	2.75000E+00	-7.92324E-01	1.34112E-01	7.37039E+00	4.17953E+00
6	3.00000E+00	-7.58765E-01	1.34340E-01	8.41497E+00	4.17723E+00
6	3.25000E+00	-7.25160E-01	1.34490E-01	9.45904E+00	4.17542E+00
6	3.50000E+00	-6.91524E-01	1.34594E-01	1.05027E+01	4.17398E+00
6	3.75000E+00	-6.57866E-01	1.34667E-01	1.15461E+01	4.17279E+00
7	4.00000E+00	-6.24192E-01	1.34720E-01	1.25891E+01	4.17179E+00

3.2 Using block 1

The first block contains the system of differential equations. These equations are processed by ATOMCC to determine the recurrence relations which should be written into ATSPGM to generate the Taylor series for each component of the solution.

- 30 -

(Internal page reference for manual.doc)

The first block may also be used for ATOMCC options to control the operation of the translator. This is done by placing a COPTION card at the beginning of ODEINP. Multiple translator options may be specified on the same line, i.e. COPTION DOUBLE, LENVAR=40. Multiple COPTION cards are also allowed, but they must precede all other cards.

3.2.1 Format for the system of equations

For the differential equations, DIFF(Y,X,N) is used to denote the n-th derivative of the dependent variable y with respect to the independent variable x. The value of the variable N may range from 1 to 6, inclusively. The DIFF(X,Y,N) function is used to specify ODE's just as with other standard FORTRAN operators and functions. All functions supported by ATOMCC are listed in Section 4.2. The statements in ODEINP can be either upper- or lower-case letters.

We use upper-case in this manual for emphasis.

The input to ATOMCC follows FORTRAN conventions. Comment cards contain a 'C' in column 1. The entire comment card is reproduced in ATSPGM. A comment card must not contain a block terminator '\$'. Columns 1-5 are used to enter line numbers. Column 6 is used for continuation; there is a limit of 19 continuation lines in FORTRAN. Columns 7-72, where the block terminator '\$' must appear, contain the statements of the equations. Columns 73-80 are ignored. As in FORTRAN, blanks are not significant. (A word of caution, the tab character is not recognized by ATOMCC.)

The equations in block 1 must be of the form

```
DIFF(X,Y,N) = expression,
variable = DIFF(X,Y,N),
or variable = expression.
```

An expression may contain operations on variables and DIFF(,,) functions. The highest order derivative of each dependent variable must be given explicitly by an equation of the form DIFF(,,) = expression, but DIFF(,,) functions of lower order may appear in expressions on the right hand side. A system of differential equations may be specified with more than one independent variable. But, each independent variable will have the same value as the solution is computed. Independent variables are implicitly defined by the DIFF(,,) function for that independent variable, so explicitly defining one in an assignment statement will cause an error message to be printed which indicates that the independent variable in question has already been defined.

Incidentally, ATOMCC transforms all constant integer powers of a variable up to 4 into multiplications. This is done in order to avoid the problems raised by the initial value of that variable being equal to zero. For integer powers greater than 4, the user is responsible to see that a l'Hopital situation does not arise.

Except for integer powers, constant coefficients which appear in ODE's are converted to real numbers by ATOMCC, so that it does not matter whether three is written as 3, 3.0, 3.E0, or 3.D0.

- 31 -
(Internal page reference for manual.doc)

3.2.2 Parameters in the equations

Example 3-1 shows a system involving parameters. It is often interesting to explore the dependence of the solution on parameters which appear in the differential equation. The ATOMCC system is especially well suited to this problem, because ATSPGM can be generated only once and then executed repeatedly for different values of the parameters.

You may enter the parameters as constants directly into the statement of the equations. In that case, ATOMCC writes those constants directly into ATSPGM; this approach does not allow easy modification of the parameters. Note that all constants (except integer powers) are converted to real numbers by ATOMCC, so that it does not matter whether three is written as 3, 3.0, 3.E0, or 3.D0.

If you wish to change the values of the parameters, make each parameter in the equation to appear as a variable as shown in Example 3-1. Note that the variable parameters must be assigned dummy values in block 1, even though the actual values may be specified at solution time by statements in block 3.

If the values of the parameters are to be supplied at solution time by reading from a data file or from a terminal, it may be convenient to solve the problem repeatedly as shown by Example 3-14 in Section 3.4.2.

3.2.3 COPTION DOUBLE - Double-precision ATSPGM

Unless you specify differently, the ATSPGM program generated by ATOMCC is written in single-precision. A COPTION DOUBLE card, as the first card in ODEINP, signals ATOMCC to write ATSPGM in double-precision. No other changes should be made in block 1. In particular, if the ODE's have library functions such as SIN, EXP, etc., ATOMCC automatically inserts DSIN, DEXP, etc. into ATSPGM. You should not make those substitutions yourself.

Example 3-7. Double precision object program.

```
COPTION DOUBLE
  DIFF(Y,T,2) = 6*Y*Y + T      $
  $
C Read initial conditions and integration interval
  OPEN(5,FILE='DATA')
  READ(5,1010) START,END,Y(1),Y(2)
1010 FORMAT(4F10.3)
  WRITE(*,1020) START,END,Y(1),Y(2)
1020 FORMAT(' Solve the first Painleve transcendent' /
+ ' Interval: ',2F10.3 /
+ ' Initial conditions:',2F10.3 /)      $
  $
```


- 32 -
(Internal page reference for manual.doc)

In a double-precision ATSPGM program:-

- real variables are declared as double precision;
- double-precision FORTRAN functions (i.e. DLOG) are generated in ATSPGM; (The user must still use single-precision functions in the first block.)
- call statements in ATSPGM reference the double-precision ATOMCC library routines DRSET, DSTEP, and DRDCV;
- FORMAT statements use D-format for real variables;
- constants are generated in double-precision form.

Some changes may be required in blocks 2, 3, and 4. ATOMCC copies them directly into ATSPGM, so you are responsible for any changes such as declarations or format modifications.

Be sure to link ATSPGM.OBJ with the double-precision ATOMCC subroutine library, DRDCV.OBJ. Incidentally, stiff problems are solved only in double-precision.

3.2.4 COPTION COMPLX - Complex ATSPGM

Including a COPTION COMPLX card as the first card in ODEINP signals ATOMCC to write a single-precision complex ATSPGM.

Example 3-8. Complex object program.

```
COPTION COMPLX
  DIFF(Y,T,2) = 6*Y*Y + T      $
  $
  OPEN(5,FILE='DATA')
  READ(5,1010) MPRINT,NSTEPS,KPTS
1010 FORMAT(3I5)
  WRITE(*,1010) MPRINT,NSTEPS,KPTS
C
C Read initial conditions
  READ(5,1020) Y(1),Y(2)
1020 FORMAT(8F10.3)
  WRITE(*,1020) Y(1),Y(2)
C
C Read piecewise linear path
  READ(5,1020) (POINTS(I),I=1,KPTS)
  WRITE(*,1020) (POINTS(I),I=1,KPTS)  $
  $
```

No other changes should be made in block 1. If the ODE's have library functions like SIN, EXP, etc., ATOMCC automatically inserts CSIN, CEXP, etc. into ATSPGM. You should not write any complex functions yourself.

Some changes may be required in blocks 2, 3, and 4. ATOMCC copies them directly into ATSPGM, so you are responsible for any

- 33 -
(Internal page reference for manual.doc)

changes such as declarations or format modifications. The independent and dependent variables are complex, so each must have both a real and an imaginary part. So, FORMAT 1010 and 1020 are changed from Example 3-7 to Example 3-8. We suggest that you study an example of a complex ATSPGM to see which variables are of type COMPLEX before you attempt to execute the program.

The other important change required for a complex ATSPGM is the manner in which the path of integration is specified. The path of integration is a piecewise linear path in the complex plane of the independent variable. The path consists of KPTS points stored in the complex array POINTS as shown in Example 3-8. The complex path of integration is discussed further in Section 3.4.14.

For a normal search of the complex plane for singularities, it is best to keep MPRINT=4. Higher values of MPRINT only leads to ever more confusing amounts of print outputs.

3.2.5 COPTION DOUBLE, COMPLX - Double-complex ATSPGM

Including a COPTION DOUBLE,COMPLX card, as the first card in ODEINP, signals ATOMCC to write a double-precision complex ATSPGM.

Example 3-9. Double complex object program.

```
COPTION DOUBLE,COMPLX
  DIFF(Y,T,2) = 6*Y*Y + T    $
  $
  OPEN(5,FILE='DATA')
  READ(5,1010) MPRINT,NSTEPS,KPTS
1010 FORMAT(3I5)
  WRITE(*,1010) MPRINT,NSTEPS,KPTS
C
C Read initial conditions
  READ(5,1020) Y(1),Y(2)
1020 FORMAT(8F10.3)
  WRITE(*,1020) Y(1),Y(2)
C
C Read piecewise linear path
  READ(5,1020) (POINTS(I),I=1,KPTS)
  WRITE(*,1020) (POINTS(I),I=1,KPTS) $
  $
```

No other changes should be made in block 1. In particular, if the ODE's have library functions like SIN, EXP, etc., ATOMCC automatically inserts CDSIN, CDEXP, etc. into ATSPGM. You should not make those substitutions yourself. Example 3-10 is a portion of ATSPGM generated for the input shown in Example 3-9.

Example 3-10. ATSPGM for Example 3-9

```
c*****
c   This program was produced by the ATOMCC translator version 7.10
c                                     Copyright (C) 1985, Y. F. Chang
c*****
```


- 34 -

(Internal page reference for manual.doc)

```

c Portions (c) Copyright, Microsoft Corp., 1981. All rights reserved.
c This program was written for the following inputs
c
COPTION DOUBLE,COMPLX
C      DIFF(Y,T,2) = 6*Y*Y + T
c-----
c no instructions in second input block
c-----
      COMMON /IPASS/ LENSER,LENVAR,MPRINT,MSTIFF,LRUN,
+ KTRDCV,KNTSTP,KTSTIF,KXPNUM,KDIGS,KENDFG,NTERMS,NOPT
A /RPASS/ RADIUS,ERRLIM,ADJSTF,RCREAL,RCIMAG,RDCERR
B /CPASS/ START,END,ORDER
C /DPASS/ H,HNEW
D /PATHCM/ POINTS,VECTOR,KPTS,KPAST
COMPLEX*16 START,END,POINTS(10),VECTOR,CZRO,DCMLPX
COMPLEX ORDER
DOUBLE PRECISION H,HNEW,AL
COMPLEX*16 TMP5( 36, 1)
CHARACTER*6 NAMES
EQUIVALENCE (TMP5(1,1),Y(1))
COMPLEX*16 Y(36), T(2), TMPAAB(30), TMPAAA(30)
COMPLEX*16 SHIFT
DIMENSION NAMES(1)
DATA NAMES(1)/'Y.....'/
Y(33) = 1.1
9 FORMAT(2X,11HAbove is at,1P2D12.4,9H step no.,I4/)
10 FORMAT(72H  ATOMCC  Ver. 7.10, Copyright (C) 1985, Y. F. Chang; S
Aolution results./9H  *****)

```

```

      CZRO = DCMLPX(0.D0,0.D0)

```

```

c-----
c start of third input block
c-----
      OPEN(5,FILE='DATA')
      READ(5,1010) MPRINT,NSTEPS,KPTS
1010 FORMAT(3I5)
      WRITE(*,1010) MPRINT,NSTEPS,KPTS
C
C Read initial conditions
      READ(5,1020) Y(1),Y(2)
1020 FORMAT(8F10.3)
      WRITE(*,1020) Y(1),Y(2)
C
C Read piecewise linear path
      READ(5,1020) (POINTS(I),I=1,KPTS)
      WRITE(*,1020) (POINTS(I),I=1,KPTS)
c-----
c end of third input block
c-----

```

- 35 -
(Internal page reference for manual.doc)

```

c-----
c Calculate radius of convergence and take optimum step.
c-----
      CALL CDRDCV(TMPS,LENVAR,NUMEQS,NAMES)
c-----
c no instructions in fourth input block
c-----
      GO TO (26,28), KENDFG
26     AL = RADIUS
      H = DSIGN(AL,H)
      IF(MPRINT.GT.4 .OR. NOPT.EQ.0) GO TO 24
      WRITE(*,9) START,KNTSTP
24     START = START + HNEW*VECTOR
      IF(MPRINT.LT.5) GO TO 27
      WRITE(*,11) KNTSTP,START
      K = Y(33)
      WRITE(*,12) NAMES(K),Y(1), Y(2)
27     CONTINUE
      WRITE(*,14) NSTEPS
28     CONTINUE
29     STOP
      END

```

Some changes may be required in blocks 2, 3, and 4. ATOMCC copies them directly into ATSPGM, so you are responsible for any changes such as declarations or format modifications. The independent and dependent variables are complex, so each initial condition must have both a real and an imaginary part.

The important change required for a complex ATSPGM is the manner in which the path of integration is specified. The path of integration is a piecewise linear path in the complex plane of the independent variable. The path consists of KPTS points stored in the complex array POINTS as shown in Example 3-8. The complex path of integration is discussed further in Section 3.4.14.

For a normal search of the complex plane for singularities, it is best to keep MPRINT=4. Higher values of MPRINT only leads to ever more confusing amounts of print outputs.

3.2.6 COPTION LENVAR=n - Series length

The size of the arrays which contain each series is stored in LENVAR. The value of LENVAR may be changed by placing the expression "LENVAR=n" on a COPTION card. The length of the series used is controlled by the variable LENSER (see Section 3.4.9).

Example 3-12. COPTION LENVAR=n

```

COPTION LENVAR=100
      DIFF(POWER,TIME,4) = 3*DP2
      DP2 = DIFF(POWER,TIME,2)      $
      $
      OPEN(5,FILE='DATA')
      READ(5,1010) START,END,(POWER(I),I=1,4)
1010  FORMAT(6F10.3)

```


- 36 -
(Internal page reference for manual.doc)

```
WRITE(*,1010) START,END,(POWER(I),I=1,4) $
$
```

There are several circumstances under which you may wish to use a series length which differs from the 30-term series used by ATSPGM:-

- The system of ODE's contains no functions or products of dependent variables. The cost of generating the series is then proportional to the series length (instead of the length squared), so using longer series may result in a faster execution time.
- A series begins with many zero terms. Since ATSPGM requires series which have at least 8 nonzero terms, you must lengthen the series.

Section 3.4.9 contains a discussion of the effect of series length on the execution time of ATSPGM.

3.2.7 COPTION DUMP=n - Diagnostic messages

This feature involves the operation of the ATOMCC translator itself and will rarely be used. It was used during the development of the software and is documented here only for completeness. The form is COPTION DUMP=n, where n indicates the amount of dump to be written (on your screen). The following is dumped for each n.

```
5 after lexical and syntactical analysis,
and 4 after first optimization,
and 3 after equation sorting and variable type identification,
and 2 after implicit operator and operand analysis,
and 1 after second optimization.
```

It is suggested that users who wish to consult with the authors about a suspected bug in ATOMCC should have available the output from a run in which "DUMP=5" was specified and directed to a file for printing.

3.3 Using block 2

The second, third, and fourth blocks are not processed by ATOMCC in the same way as the first input block. ATOMCC merely copies the data from your ODEINP file directly into ATSPGM.

The second block is optional and is used to insert non-executable FORTRAN statements at the beginning of ATSPGM. This block gives the user the ability to insert 'SUBROUTINE', 'FUNCTION', 'COMMON', 'DATA', and 'DIMENSION' into ATSPGM. Example 3-13 shows where statements supplied in block 2 are inserted into ATSPGM.

3.3.1 Subroutine form of ATSPGM

ATSPGM is normally generated as a main program which can then be run to solve the specified problem. The ATOMCC translator has the capability to generate subroutine or function forms for ATSPGM. The subroutine or function can then be called from a user supplied main program, or from another ATOMCC-generated main program, to obtain the solution of the problem.

A subroutine ATSPGM is specified by placing a SUBROUTINE statement in the second block. This statement is placed unchanged at the beginning of the generated code, so you have complete control of the name of the subroutine and its parameter list. However, this also means that you have complete responsibility for passing values to and returning values from the generated program segment. ATSPGM written by ATOMCC will have a RETURN statement instead of a STOP at its end. Example 3-13 shows the ODEINP for a subroutine ATSPGM named DIFEQU, which solves the equation $f' = \log(\sin(x) + f)$.

Example 3-13. Subroutine form of ATSPGM

```
DIFF(F,X,1) = ALOG(SIN(X) + F)  $
SUBROUTINE DIFEQU(COND)         $
F(1) = COND $
$
```

with its calling program

C Driver program for Example 3-13.

C

C A very simple subroutine, START and END are passed through COMMON.

C

```
COMMON/CPASS/ START,END,ORDER
1010 FORMAT(6F10.3)
READ(5,1010) COND,START,END
WRITE(*,1010) COND,START,END
CALL DIFEQU(COND)
STOP
END
```

Many of the variables used in ATSPGM appear in COMMON, so they cannot be passed as parameters. Values can be passed through the parameter list as temporary variables, or they can be passed in COMMON by including the appropriate COMMON block in the calling program as illustrated by Example 3-13.

3.3.2 User declarations

You may need to declare the type of the additional variables you introduce into ATSPGM, if you use the double-precision or complex forms of ATSPGM. You may include appropriate DIMENSION, DOUBLE, COMMON, DATA, etc. in block 2. Any non-executable FORTRAN statements inserted in the second block are copied into the

- 38 -

(Internal page reference for manual.doc)

beginning of ATSPGM exactly as they are written, so you have total control over the statements entered. These statements must conform to FORTRAN specifications; ATOMCC does not check their syntax or the order of placement.

3.3.3 Common blocks for user

You may wish to use COMMON blocks. COMMON declarations may be included in block 2 like all non-executable FORTRAN statements discussed in Section 3.3.2.

Note that many of the variables used in ATSPGM appear in COMMON blocks and must not be assigned to other blocks.

3.4 Using block 3

The third block must be used to specify the interval of integration and the initial conditions. It may also be used to change the default values of method-control variables. Default values are changed by inserting statements which assign new values to these variables. If desired, other statements can be inserted into ATSPGM at this point. The user should have an understanding of the form of ATSPGM and the relative position of the third block in ATSPGM. Examples 2-1 and 2-3 and Examples 3-1 and 3-3 are a pair of ODEINP files with their respective ATSPGM programs. you can better understand how the method-control variables work by studying the statements in the neighborhood of block 3.

The assignment of values to each of the variables discussed in this Section may be done by:

1. a READ statement (see Examples 2-1, 2-2, 3-1, 3-7, 3-11, 3-12),
2. an assignment statement (see Example 2-6), or
3. being passed as temporary variables through a parameter list (see Example 3-13), or
4. common block shared with a driving main program (see Example 3-13).

- 39 -
(Internal page reference for manual.doc)

3.4.1 Initial conditions

The initial conditions must be specified in the third block. The initial values at $x_0 = \text{START}$ of the dependent variable, say yyy , and its derivatives are assigned to the array yyy as follows:-

$YYY(1)$ denotes yyy at x_0 ,
 $YYY(2)$ denotes yyy' at x_0 ,
 $YYY(3)$ denotes yyy'' at x_0 , etc.

The number of initial conditions which must be specified for each dependent variable equals the highest derivative of that variable which appears in the system of equations. If the system includes a $\text{DIFF}(y,x,5)$, then five initial conditions must be included for the dependent variable y . ATOMCC does not check whether initial conditions have been supplied, since it is possible that the series variable has been passed through a subroutine parameter list. Therefore, it is your responsibility to see that the required initial conditions are defined.

When ATSPGM is a subroutine, the series variable may be passed in the parameter list. In this case, the initial conditions may be stored in the proper elements of the series array before ATSPGM is executed. An alternative to passing the series via the parameter list is to pass the initial conditions as temporary variables in the parameter list. Assignment statements in the third block then assign the values to the appropriate dependent variable array elements.

Note to those who are familiar with the Taylor series method:- the initial conditions should be entered as explained above; that is, $Y(3) = y''(x_0)$, not $y''(x_0)*h*h/2$, since this shifting is done automatically by ATSPGM .

3.4.2 Parameters in the differential equations

It is often interesting to explore the dependence of the solution on parameters in the ODE's. Section 3.2.2 and Example 3-1 showed how the system of equations is entered and how values of the parameter are assigned. A refined version of Example 3-1 is shown in Example 3-14. Here, a loop is used to read successive values of the parameter. See the object program listing in Example 3-3 and observe that statement '28 CONTINUE' is provided by ATOMCC for this purpose.

Example 3-14. Read parameters repeatedly.

```
DIFF(X,T,2) = - ALPHA*X*R
DIFF(Y,T,2) = - ALPHA*Y*R
R = (X*X + Y*Y)**(-1.5)
ALPHA = 0.65      $
$
C Read and echo print code, number of integration steps.
OPEN(5,FILE='DATA')
```


- 40 -
(Internal page reference for manual.doc)

```

      READ(5,1010) MPRINT,NSTEPS
1010  FORMAT(2I10)
      WRITE(*,1010) MPRINT,NSTEPS
C
C Read and echo interval and initial conditions to be used each time.
      READ(5,1020) OLDSTR,OLDEND,X1,X2,Y1,Y2
1020  FORMAT(6F10.3)
      WRITE(*,1020) OLDSTR,OLDEND,X1,X2,Y1,Y2
C
C Loop for different values of the parameter.
      DO 28 IPROB=1,20
      READ(5,1020) ALPHA
      IF(ALPHA.EQ.0.0) STOP
      WRITE(*,1020) ALPHA
      START = OLDSTR
      END = OLDEND
      X(1) = X1
      X(2) = X2
      Y(1) = Y1
      Y(2) = Y2
      WRITE(7,1030) KINTS,START,X(1),X(2),Y(1),Y(2)
1030  FORMAT(I5,1P5E14.5)

```

3.4.3 Solve a problem repeatedly

In Example 3-14, the statement 'DO 28 IPROB=1,20' is included in block 3 to solve the same system of differential equations as many as 20 times without restarting the program. The statement '28 CONTINUE' near the end of ATSPGM is written by ATOMCC for this purpose. Within this loop, you may vary values of parameters as in this example, you may vary the initial conditions, or you may vary the method-control variables.

3.4.4 START, END - Interval of integration

The integration interval must be specified in the third block using the two reserved words START and END. Their use is illustrated by all the examples.

If a complex ATSPGM is being used, then the interval of integration is a piecewise linear path in the complex plane of the independent variable. The specification of complex paths of integration is discussed in Sections 3.4.13 and 3.4.14.

3.4.5 NSTEPS - Number of integration steps, default=40

The maximum number of integration steps taken during the solution is the variable NSTEPS. The default value (40) can be so small because the use of long series allows very large integration steps. You may change this value by inserting a statement in the third input block which assigns a new value to NSTEPS, or you may read in a value for NSTEPS from a data file.

For some stiff problems and in searching for singularities in the complex plane, it is best to set NSTEPS at a large value.

3.4.6 H - Initial trial stepsize, default = 1.0

The default value of the suggested initial stepsize can be changed by assigning the desired value to the variable H in the third block. The term "suggested initial stepsize" is used because this value may be adjusted by ATSPGM before the first step is taken. This adjustment is made if its need is indicated by an analysis of the Taylor series for underflow/overflow. The user can rarely make a better choice than ATOMCC.

For stiff problems, in addition to the adjustment of H in ATSPGM at the first integration step, the stepsize H is adjusted within DRDCV at every step. The adjustments at the first integration step may need some manual assistance. In such cases, observe the values generated by the ATSPGM program and project forward to the next reasonable value and enter it for H.

3.4.7 ERR LIM - Preset accuracy of the solution

The local error tolerance is the variable ERR LIM. ATSPGM will keep the maximum local error less than ERR LIM. The magnitude of ERR LIM is automatically set by ATOMCC to be close to the computer round-off error in both single- and double-precision. You may set ERR LIM to be much larger; however, your results will become inaccurate. You may not set ERR LIM to be much smaller.

The error analysis for normal problems is so precise that one can almost expect the global error to be under control. This is one numerical method where the global error is proportional to the local error. Therefore, to determine the magnitude of the global error, one only needs to run the solution a second time with an ERR LIM set one order of magnitude smaller than the first run and compare the two results.

3.4.8 ADJUSTF - Error control for stiff problems

The error analysis for stiff problems is not nearly as well developed as the error analysis for normal problems. In solving stiff problems, it is necessary to make assumptions regarding both the exponential function and the polynomial that are used to fit the solution being sought. Therefore, an error-controlling parameter separate from ERR LIM is used. The default value for ADJUSTF is 1.E-2. Also, the value of ERR LIM is fixed to 1.E-6 for stiff problems. The user is advised to change ADJUSTF to lower values and make additional runs of the solution to be certain of its correctness.

There is absolutely no connection between a particular value of ADJUSTF and the magnitude of the error in the computations. The most that is claimed is that as the value of ADJUSTF is adjusted smaller, there is likely to be a lowering of the computational error. Since the nature of stiff solutions is an approximation, there can be no true error control. And so, none is attempted.

3.4.9 LENSER - Length of series used, default=30

The length of the Taylor series is the variable LENSER. This value may be changed by assigning a new value to LENSER in the third block. However, there are some restrictions governing how this is done.

LENSER may be set to any integer between 15 and 30 without any other changes. Series of fewer than 15 terms should not be used. If a series of more than 30 terms is desired, the size of all series variables (LENVAR) must be increased (see Section 3.2.6). The user is reminded that the execution time is related to the length of the series. The default value of LENSER=30 have been found to be optimal for fast execution.

In stiff solutions, LENSER is automatically set to either 15 or 10 depending on the parameter MSTIFF. In these cases, the user cannot change these set values of LENSER without going into the subroutine DRDCV.

3.4.10 MPRINT - Amount of print produced, default=4

The amount of printout produced by ATSPGM during the solution of the problem is controlled by the variable MPRINT. Values of every dependent variable at each integration step is printed with an MPRINT=4. The user can change the default by assigning MPRINT a different value in the third block. The amount of printout produced for values of MPRINT is listed below.

- 0 Used for timing purposes; printout is produced only when a fatal error occurs.
- 1 No print is produced, but the loop controlled by RSET is activated to produce user controlled print (see Section 3.5.2).
- 2 Print is produced only at points selected by the user. The printout consists of the integration step number, the value of the independent variable and the initial conditions for each dependent variable.
- 4 (default) Print the information for 2 at every integration step. (In complex solutions, only the singularity locations are printed here.)
- 5 In addition to the output under 4, the actual stepsize used at each integration step is printed. In stiff problems, the exponential function and the length of the polynomial are printed. In stiff problems, the estimated HSTF and the relative goodness of the exponential fit are printed. (In complex solutions, MPRINT=5 is equivalent to 4)
- 6 In addition to the information for 5, print (a)the computed radius of convergence, (b)location of the singularity(ies), and (c)which test was used to locate the poles.

(Internal page reference for manual.doc)

- 7 In stiff problems, the entire results for the exponential fit are printed.
- 8 All of MPRINT=6, plus (a) the estimated error in h/Rc , (b) the values of the elements of each series, and (c) messages for all tests used for the radius of convergence.
- 10 All diagnostic messages including all the series terms.

Users who will use ATOMCC often should try setting MPRINT=10 to become familiar with the information available during the solution of a problem.

The value of MPRINT may be dynamically controlled during the computation of a solution by inserting statements in the fourth block which test the current value of START (or KINTS), and set the value of MPRINT accordingly.

3.4.11 DLTXPT - Print point increments, default = 0.0

ATSPGM uses variables XPRINT and DLTXPT to control values of the independent variable for which print is produced. We will discuss a variety of ways these variables can be used.

If DLTXPT=0.0, then print is produced only at the integration steps chosen by the program. This is the default condition.

Print is produced at equally spaced points by specifying DLTXPT = xxx (the desired spacing) as shown in Example 2-8 in Section 2.3. A statement may be placed in the third block which assigns the desired value to DLTXPT. You should also specify MPRINT=2. Otherwise print is produced both at your selected points and also at the integration steps selected by ATSPGM. MPRINT is discussed in Section 3.4.10.

The values of each dependent variable are printed at the equally spaced points by expanding a series which has already been computed. The integration does not step to the equally spaced points. Hence, requesting intermediate output has no effect on the number or size of integration steps taken.

More creative print points are possible by the use of block 4. See Example 3-15 in Section 3.5.3.

DLTXPT can be used in stiff solutions to generate output at desired values of the independent variable. DLTXPT cannot be used in conjunction with ZEROT, where the solution may be stopped for any value of any function.

3.4.12 KTRDCV - Dynamic suppression of CALL RDCV

KTRDCV can be used to speed up the execution of ATSPGM for systems of ODE's for which some components of the solution do not constrain the integration stepsize. For KTRDCV=N, the series analysis is performed only for the first N components of the solution. You can generate ATSPGM once, run it to see the radii of

- 44 -
(Internal page reference for manual.doc)

convergence of each component, and use KTRDCV on subsequent runs to reduce the execution time of ATSPGM. You should order the input ODE's such that those with larger radii are listed last. Careful study of the ATSPGM program for a sample system of ODE's will help you see how KTRDCV works.

3.4.13 KPTS - Number of points on complex path

In Section 3.4.4, we discussed the specification of the interval of integration using START and END. If ATOMCC has been directed to generate complex object code (COPTION COMPLX, Sections 3.2.4 or 3.4.5), then the path of integration is a piecewise linear path in the complex plane of the independent variable. See Example 3-8 in Section 3.2.4.

The variable KPTS is the number of vertices belonging to that piecewise linear path, including both of the endpoints. The complex array 'POINTS' holds the vertices. POINTS(1) becomes START, and POINTS(KPTS) becomes END. The value of the solution is printed at each element of POINTS. KPTS may be at most 10.

3.4.14 POINTS - Complex path of integration

The complex array 'POINTS' specifies the path of integration in the complex plane of the independent variable. Its use with KPTS is discussed in Section 3.4.13. There may be at most 10 points specified.

3.4.15 MSTIFF=10 - Solutions which are entire

Solutions which are entire (have no singularities in the finite plane), should not be solved using the ATOMCC system. It is a total waste of computing power to solve linear problems using ATOMCC. This is particularly true for linear 'stiff' problems.

It can be EASILY show that ALL solutions that are entire can be solved in quasi-closed forms. This INCLUDES two-point boundary value problems!

If ATOMCC recognizes that a system of ODE's involves no functions or products (an entire solution), it sets MSTIFF=10 in ATSPGM. Subroutine RDCV recognizes MSTIFF=10 as a flag for a special test for series that is entire.

3.4.16 MSTIFF=20,21,22 - Stiff problems.

This version of ATOMCC contains a double-precision algorithm to solve stiff problems. To use it, one can either set MSTIFF=20, or 21, or 22. Other parameters that should be controlled are H, ADJUSTF, and NSTEPS. It is also desirable to set MPRINT to 7, at least initially, for observing the progress of the solution. If it should be evident that the problem is not really stiff, then it is most advisable to solve it as a normal problem.

- 45 -
(Internal page reference for manual.doc)

MSTIFF=20 is the more conservative of the three algorithms. In this case, LENSER is set to be 15. The default value for ADJSTF, the error-controlling parameter, is a rather large 1.E-2. The user should run the stiff solution at least one more time with a somewhat smaller ADJSTF, say 1.E-3, to check on its validity. Since the default value for NSTEPS is 40, this should definitely be set at 500 or more, to be sure that the stiff solution can be completed. The initial stepsize H may need some adjustment by the user. He should study the H-adjustment messages from ATSPGM and take over control if and when the automatic adjustment is incapable of reaching a desirable value for H. When MSTIFF=20, those stiff problems that have steady-state solutions are identified. After which, one should perform some manual manipulations before re-submitting the problem to ATOMCC.

When MSTIFF=21, LENSER is set to only 10. So, this option should be used only if the user is absolutely certain that the problem under study is very stiff. The solution of stiff problems under this option is considerably faster than that for MSTIFF=20, because not only are the series shorter, the integration stepsizes are considerably larger. The same statements as above applies for the parameters H, ADJSTF, and NSTEPS. (One of the Enright-Hull set of stiff problems, E2, was solved by ATOMCC in one step!) There is no attempt to identify steady-state solutions when MSTIFF=21.

MSTIFF=22 is identical to MSTIFF=20 except for the fact that there is no attempt to identify steady-state solutions.

As mentioned above, the stiff algorithm is written in double-precision. It is simply not cost effective to solve such problems using single-precision. There is one other restriction on stiff problems. All such problems must be stated as first-degree ODE's.

The regular printing option, DLTXPT, functions properly in stiff solutions. So, it is possible to obtain uniformly spaced print points or logarithmically spaced print points.

3.4.16.1 Steady-State Stiff Problems

There are some stiff problems that approach steady-state solutions. Sometimes this occurs with all of the functions becoming constant simultaneously. In such a case, the ATOMCC solution will stop and points out the fact that every function seem to have reached constant values. The user can decrease ADJSTF and repeat the solution to verify the truth of this fact. He can also run the problem with MSTIFF=22 and observe the perpetual singleness of the results.

In other instances, a particular function reaches constancy before any of the others. When this happens, the following message will be printed, with obvious meaning.

```
The function Y4.... is constant at 3.115283D-04
Look for a steady-state solution.
Set all derivatives to zero, use the value of Y4.... given above,
and solve for the other functions in easy situations.
Then, re-submit to ATOMCC. Use MSTIFF=21 or 22, do not use MSTIFF=20.
```


- 46 -
(Internal page reference for manual.doc)

This example is for the problem CHEM6 out of the Enright-Hull set of chemical stiff problems. This solution stopped at time = 0.0561, where the solution had hardly gotten started. After solving the easy cases as suggested, the ATOMCC solution of the re-submitted problem reached the final time of 1000 in 26 steps using MSTIFF=21!

The equational input for the first ATOMCC run is as follows.

```
DIFF(Y1,T,1) = 1.3*(Y3-Y1) + 10400*AK*Y2
DIFF(Y2,T,1) = 1880*(Y4 - Y2*(1+AK))
DIFF(Y3,T,1) = 1752 - 269*Y3 + 267*Y1
DIFF(Y4,T,1) = 0.1 + 320*Y2 - 321*Y4
AK = EXP(20.7 - 1500/Y1)
```

The equational input for the second ATOMCC run, after encountering the constancy message is as follows.

```
DIFF(Y1,T,1) = 1.3*(Y3-Y1) + 10400*AK*Y2
DIFF(Y2,T,1) = 1880*(Y4 - Y2*(1+AK))
Y3 = 1752/269 + 267/269*Y1
Y4 = 3.115283E-04
AK = EXP(20.7 - 1500/Y1)
```

The differences between to two inputs are in the third and fourth equations.

3.5 Using block 4

ATOMCC copies the fourth block directly into ATSPGM near the end of the code for each integration step. Example 3-3 in Section 3.1.3 shows the location of block 4 in ATSPGM. The fourth block is used to tailor ATSPGM to your special requirements. Several examples are provided in Section 3.1.1, but many other creative uses are possible. Most applications require a good knowledge of the ATOMCC system for solving ODE's.

3.5.1 Automatic printing of output points

The ATSPGM program automatically prints the values of each component of the solution at each integration step (see Example 2-7 in Section 2.2.6). It can print the same information at equally spaced points you select using DLTXPT (see Section 3.4.11). This technique for generating output at equally spaced points requires no use of block 4, but it will help you to understand subsequent sections if you understand how ATSPGM generates this equally spaced output. Please refer to the object code in Example 3-3 in Section 3.1.3 as you read.

The points selected for output do not affect the integration steps so that the local error remains proportional to the global error. For each output point within an integration step, the

- 47 -
(Internal page reference for manual.doc)

solution is computed by evaluating its Taylor series. Inside the subroutine RSET, each output point within the the integration step is controlled by a loop which begins with the statement '24 IF (KENDFG.EQ.3) KENDFG=1' and ends with the statement 'GO TO (26,28,24), KENDFG'. KENDFG is a flag which controls the loop.

Subroutine RDCV sets KENDFG=2 if the current steps reaches END, otherwise KENDFG=1. Subroutine RSET determines whether the next output point (XPRINT) lies within the next step of integration. If not, RSET leaves KENDFG unchanged (1 or 2) and stores the initial conditions required by the next step. If there is a print point within the next integration step, RSET prints the solution at that output point (suppressed if MPRINT=0 or 1), stores values of the series and its derivatives as elements LENSER+1, LENSER+2,... in the series, and sets KENDFG=3. Then, the 'GO TO' in ATSPGM returns to label 24 to handle additional output points within the next integration step. Once all of the output points within the next step have been passed, RSET returns KENDFG to its original value (1 or 2), and the solution proceeds forward.

You can use DLTXT to generate output at equally spaced points without understanding how that output is generated, but if you wish to produce some special output as discussed in the following Sections, this understanding is necessary.

3.5.2 User controlled printing of output points

The object program automatically prints the values of each component of the solution at each integration step (see Example 2-7 in Section 2.2.6). The method used to produce output at user selected points is discussed in Section 3.5.2. However, you may use block 4 as shown in Example 3-1 in Section 3.1.1 to produce output according to your particular needs.

The execution time for ATSPGM is sensitive to the amount of output produced. Unless you are interested in the automatically produced output, place MPRINT=1 in block 3. Then, RSET controls the necessary looping as discussed in Section 3.5.1, but it produces no output. This yields the solution at equally spaced points. Unequal spacing can be achieved by changing DLTXT within block 4. Example 3-15 in Section 3.5.3 shows how to obtain the solution at logarithmically spaced points.

Many variables are accessible for your use in block 4. The elements of the array for each dependent variable, [y(LENSER+1), y(LENSER+2), etc.], contain the values of that variable and its derivatives evaluated at XPRINT. The user can therefore print these values totally independent from the print produced by ATSPGM. Additional usable values are stored in the top three positions of the series; y(LENVAR-2) is the series length actually used for that particular variable, y(LENVAR-1) is the multiplying constant for the exponential function in stiff solutions, and y(LENVAR) is the exponential coefficient in stiff solutions.

- 48 -

(Internal page reference for manual.doc)

3.5.3 Logarithmic spacing of output points

The techniques discussed in Sections 3.5.1 and 3.5.2 produce values of the solution at equally spaced points. Output at non-uniformly spaced points is obtained by adjusting DLTXT and XPRT3 within block 4 as shown by Example 3-15. The print points are at $r = 100, 50, 20, 10, 5, 2$, etc. This example also shows that problems can be integrated in the negative direction.

Example 3-15. Logarithmic print.

```

DIFF(F,R,2) = (F**3 - F)/R**2 $
$
START = 100.0
END = 1.0E-8
F(1) = 0.99
F(2) = 1.0E-4
DLTXPT = -50.0
NPTR = 1
$
IF(KENDFG .NE. 3) GO TO 25
MPRINT = 2
GO TO (501,503,502), NPTR
501 DLTXT = - 0.6*XPRT
GO TO 504
502 NPTR = 0
503 DLTXT = - 0.5*XPRT
504 NPTR = NPTR + 1
XPRT3 = XPRT + DLTXT $

```

3.5.4 ZEROT - Stopping and printing at roots of variables

It is often of interest to locate points at which a component of the solution has a root or assumes some specified value. The subroutine ZEROT, in the ATOMCC subroutine library in single-precision, do automatically solve such problems. DZEROT is the double-precision version.

The form of the CALL is

```
CALL ZEROT(NUMBER,Y,ROOT,KEY,TMPS,LENVAR,NUMEQS)
```

where

NUMBER is the index of the Y series term whose root is sought,
 Y is the variable whose root is desired,
 ROOT is the value Y is to assume (= 0 for a root),
 KEY is 1 if Y is a dependent variable, or
 0 if Y is not a dependent variable.

The arguments TMPS, LENVAR, and NUMEQS must be exactly as written above.

- 49 -
(Internal page reference for manual.doc)

Example 3-16. Rootfinding with ZEROT.

```
DIFF(Y,T,2) = 6*Y*Y + T $
$
START = 0.0
END = 1.15
ROOT = 20.0
Y(1) = 1.0
Y(2) = 0.0
$
CALL ZEROT(1,Y,ROOT,1)
$
```

When the variable whose root is being sought is not a dependent variable, the following statements are used. Note that KEY is set to 0.

```
CALL ZEROT(2,VARY,0.0,0)
IF(LRUN.NE.0) GO TO 25
TEMP = START + HNEW
WRITE(7,1010) KINTS,TEMP,VARY(1),VARY(2)
1010 FORMAT(I5,3F10.4)
GO TO 25 $
```

In these examples, it is not necessary for one to print the information as shown in the second case. The ATSPGM program does stop and restart the solution automatically at the exact root and the output is controlled by MPRINT.

ZEROT requires that the series for the variable be convergent over the range of the integration stepsize being used. If the series is not convergent over this range, then ZEROT cannot possibly locate the root with any degree of accuracy. There are two instances where this convergence requirement is not met. One is where the printing steps have been placed under the control of DLTXP, and the other is where a stiff problem is being solved. Therefore, ZEROT cannot be used for a non-zero DLTXP, and in stiff problems.

The index NUMBER can be set at any positive (non-zero) integer value; however, obviously when NUMBER is very large the accuracy of the root will suffer.

3.5.5 Finding singularities in real solutions

(This information is only for those users who are interested in finding the conjugate pairs of singularities closest to the real axis. For a more detailed study of singularities, the user should invoke COPTION COMPLEX, Section 3.2.4)

A unique feature of the ATOMCC system is its ability to provide analytic information about the solution. For example, subroutine RDCV estimates the location and order of primary singularities at each integration step in order to compute the optimal stepsize. This information may be printed using MPRINT=6 (see Section

- 50 -

(Internal page reference for manual.doc)

3.4.10). However, several steps are usually required to pass between a conjugate pair of singularities, so several different estimates for each location are written. Estimates made close to the singularities are more accurate than estimates made from further away.

Example 3-17 prints only one estimate for the location and order of each conjugate pair of singularities. That estimate is written on the step which has just begun to recede from the estimated location. Hence that step is one of the two steps closest to the singularity pair.

Example 3-17. Finding singularities in real solutions

```
COPTION DOUBLE
C
C Test dynamic printing of singularity positions.
C
  DIFF(Y1,T,1) = 2*(Y1 - Y1*Y2)
  DIFF(Y2,T,1) = Y1*Y2 - Y2      $
  $
  ERRLIM = 1.0E-6
  START = 0.0
  END = 20.0
  Y1(1) = 1.0
  Y2(1) = 3.0
  MPRINT = 10
  PRTSNG = START
  SNGTOL = 0.1
  WRITE(8,2010)
2010 FORMAT(///6H Step,9X,1Hx,10X,2HRc,8X,4HReal,
  A 8X,4HImag,7X,5Horder,7X,5Herror/)
  $
C
C Print locations of singularities.
C
  WRITE(*,2010)
  WRITE(*,2020) KINTS,START,RADIUS,RCREAL,RCIMAG,ORDER,RDCERR
C
C If this is a user print step, then continue.
  IF(KENDFG.EQ.3) GO TO 25
  IF(KINTS.EQ.NSTEPS) GO TO 102
C If this is a normal step, then jump.
  IF(KENDFG.EQ.1) GO TO 105
C Handle the last step - we might be approaching a singularity
C if we have already printed this primary singularity, then continue.
  102 IF(ABS(RCREAL-PRTSNG).LT.SNGTOL) GO TO 25
C If RDCV was uncertain, then continue.
  IF(RDCERR.GE.1.0) GO TO 25
  WRITE(8,2020) KINTS,START,RADIUS,RCREAL,RCIMAG,ORDER,RDCERR
2020 FORMAT(I4,1P6E12.3)
  GO TO 25
C
C Handle a normal step.
C
C If confused, then continue.
  105 IF(RDCERR.GE.1.0) GO TO 25
```

- 51 -
(Internal page reference for manual.doc)

```

C If approaching the primary singularity, then continue.
  IF((START-RCREAL)*H.LE.0.0) GO TO 25
C If already printed, then continue.
  IF(ABS(RCREAL-PRTSNG).LT.SNGTOL) GO TO 25
  WRITE(8,2020) KINTS,START,RADIUS,RCREAL,RCIMAG,ORDER,RDCERR
  PRTSNG = RCREAL  $

```

Example 3-18. Printed locations

Step	x	Rc	Real	Imag	order	error
1	0.000E+00	5.978E-01	-3.364E-01	4.941E-01	9.457E-01	5.588E-04
9	5.180E+00	4.954E-01	5.151E+00	4.945E-01	9.784E-01	7.081E-04
19	1.083E+01	5.297E-01	1.063E+01	4.943E-01	9.595E-01	2.888E-04
28	1.627E+01	5.145E-01	1.612E+01	4.943E-01	9.662E-01	2.660E-04

3.5.6 Stopping short of a singularity

If one component of the solution has a singularity inside the interval of integration, then the problem does not have a solution on that interval, so the integration process should stop. ATSPGM stops when NSTEPS steps have been taken (see Section 3.4.5), or it stops when the integration stepsize is so small relative to the current point of expansion that the solution would not advance.

3.6 Editing of ATSPGM

While the use of blocks 2, 3, and 4 to insert FORTRAN code directly into ATSPGM gives you a very powerful and flexible tool, you may have needs which can only be met by editing ATSPGM yourself. Section 3.6.1 uses the relatively common problem of producing efficient output at specified points to illustrate the approach.

3.6.1 TERM - Fast generation of printing at output points

The printing of solution values at equally spaced points has already been discussed in Sections 2.3, 3.4.11, 3.5.1, and 3.5.2. The technique discussed in Section 3.5.2 can be used to generate nearly all of the information about the solution at any points you select. The execution time is much faster using MPRINT=1, to suppress printing by RSET, than with MPRINT=2 or 4.

The execution time of ATSPGM can be further reduced by controlling the looping for each print point within the current integration step by calling the subroutine TERM to evaluate the series for the solution at any desired point. This subroutine is listed as Example 3-19.

- 52 -
(Internal page reference for manual.doc)

Example 3-19. TERM source listing

```

C SUBROUTINE TERM(SERIES, NTERMS, T, WORKAR)
C
C ATOMCC-LIBRARY, COPYRIGHT (C) 1983.
C
C EVALUATES SERIES AND ITS DERIVATIVES AT T
C
C AT ENTRY, 'SERIES' CONTAINS A SERIES EXPANDED AT SOME POINT
C 'START' WITH A STEP OF 'H'.
C
C AT EXIT, WORKAR(1) = FUNCTION EVALUATED AT T
C          WORKAR(2) = DERIVATIVE OF FUNCTION EVALUATED AT T
C
C          WORKAR(NTERMS) = NTERMS - 1 DERIVATIVE EVALUATED AT T
C
C PARAMETERS:
C   SERIES - SERIES BEING SUMMED
C   NTERMS - NUMBER OF INITIAL CONDITIONS NEEDED
C   T      - POINT AT WHICH SERIES IS TO BE EVALUATED
C   WORKAR - WORK AREA, SAME TYPE AS SERIES, DIMENSION NTERMS
C
C   SUBROUTINE TERM (SERIES, NTERMS, T, WORKAR)
C***** DATA DECLARATIONS
C   DIMENSION SERIES(1), WORKAR(1)
C   COMMON /CPASS/START,END,ORDER
C   A /DPASS/H,HNEW,XPRINT,DLTXPT
C   + /IPASS/LENSER,LENVAR,MPRINT,MSTIFF,LRUN,KTRDCV,
C   + INTSTP,KTSTIF,KXPNUM,KDIGS,KENDFG,NTERMS
C***** COMPUTE INITIAL CONDITIONS AND STORE THEM IN WORK AREA
C   RATIO = (T-START)/H
C   DUMMY = H/FLOAT(LENSER)
C   DO 40 I=1,NTERMS
C     ILEN = LENSER - I - 1
C     DUMMY = DUMMY*FLOAT(LENSER - I + 1)/H
C     DLOOP = DUMMY
C     SUM = SERIES(LENSER)*RATIO*DLOOP
C     DO 30 J=1,ILEN
C       LMJ = LENSER - J
C       DLOOP = DLOOP*FLOAT(LMJ-I+1)/FLOAT(LMJ)
C       SUM = (SUM + DLOOP*SERIES(LMJ))*RATIO
C   30 CONTINUE
C   WORKAR(I) = SUM + DLOOP*SERIES(I)/FLOAT(I)
C   40 CONTINUE
C***** RETURN
C   9999 RETURN
C   END

```

To illustrate the use of subroutine TERM as an example of editing ATSPGM, Example 3-20 shows the ODEINP file which was used to generate ATSPGM listed as Example 3-21. Three arrays have been dimensioned to hold the results computed by TERM, and new variables DXP and FIRSTX have been defined to fill roles analogous to DLTXPT and XPRINT.

- 53 -
(Internal page reference for manual.doc)

Example 3-20. ODEINP for TERM example

```

S = 10.0
DIFF(X,T,3) = Y - E*S*X
DIFF(Y,T,3) = - X*Z + X - E*Y
DIFF(Z,T,3) = X*Y - E*B*Z
E = 0.02
B = 8.0/3.0 $
DIMENSION USER1(3),USER2(3),USER3(3) $
OPEN(7,FILE='DATA')
OPEN(8,FILE='FAST',STATUS='NEW')
READ(7,*) X(1),Y(1),Z(1)
READ(7,*) X(2),Y(2),Z(2)
READ(7,*) X(3),Y(3),Z(3)
READ(7,*) START,END,DLTXPT
READ(7,*) DXP
FIRSTX = START
WRITE(*,1010) X(1),Y(1),Z(1),START,END,DLTXPT,FIRSTX,END,DXP
1010 FORMAT(3F10.4)
NSTEPS = 5
WRITE(8,1020)
1020 FORMAT(3X,1Ht,5X,1Hx,7X,2Hx',6X,3Hx'',
+ 5X,1Hy,7X,2Hy',6X,3Hy'',5X,1Hz,7X,2Hz',6X,3Hz''/) $
$

```

with data file

```

1.0,3.0,5.0
0.0,0.0,0.0
-1.0,0.0,1.0
0.0,5.0,0.5
0.5

```

The following lines of code must be inserted just below the CALL RDCV statement inside ATSPGM. A portion of the results are shown in Example 3-22.

Example 3-21. Object program for Example 3-20

```

CALL RDCV(TMPS,LENVAR,NUMEQS)
c===== User must insert these lines by editing.
100 IF(FIRSTX.GE.START+HNEW) GO TO 24
CALL TERM(X,3,FIRSTX,USER1)
CALL TERM(Y,3,FIRSTX,USER2)
CALL TERM(Z,3,FIRSTX,USER3)
WRITE(8,1030) FIRSTX,USER1,USER2,USER3
1030 FORMAT(1X,F3.1,1P9E8.1)
FIRSTX = FIRSTX + DXP
GO TO 100
c===== end of user inserted lines.
24 IF(KENDFG.EQ.3) KENDFG = 1

```


- 54 -
(Internal page reference for manual.doc)

Example 3-22. Output for Example 3-20

t	x	x'	x''	y	y'	y''	z	z'	z''
0.0	1.0E+00	0.0E+00	-1.0E+00	3.0E+00	0.0E+00	0.0E+00	5.0E+00	0.0E+00	1.0E+00
0.5	9.3E-01	-1.5E-01	3.9E-01	2.9E+00	-5.0E-01	-2.0E+00	5.1E+00	8.3E-01	2.3E+00
1.0	9.6E-01	3.7E-01	1.6E+00	2.3E+00	-2.0E+00	-4.1E+00	5.9E+00	2.3E+00	3.4E+00
1.5	1.4E+00	1.4E+00	2.3E+00	6.9E-01	-4.8E+00	-7.4E+00	7.5E+00	4.2E+00	4.1E+00

3.7 Large systems

As supplied to you, the ATOMCC translator can handle up to 900 equations. If you should have a need to increase this limit, a special program can be easily prepared.

3.8 Solving ODE's in the complex domain

The ATOMCC compiler allows for the solution of ODE's in the complex domain. This unique capability can be used to explore the structure of the singularities in the complex domain of non-linear problems. Linear problems, of course, have entire solution functions and therefore do not have any singularities of interest in the finite complex plane. Non-linear problems may have singularities which cover the entire complex plane.

There are essentially two types of non-linear problems; those with definite limit cycles, and those with strange attractors. For the former, the singularities form a regular lattice in the complex plane. For the latter, the singularities form structures that defy simple descriptions. The purpose of solving ODE's in the complex domain is to study the structures formed by the singularities. The ATOMCC compiler is well suited for this task, and it is the only method extant that is capable of calculating the precise location and order of all the singularities in a finite region of the complex plane of an ODE solution.

It is simple to use the ATOMCC compiler to search for the singularities. First, you must insert a COPTION COMPLX card as the first card in ODEINP. This will cause the ATOMCC compiler to generate ATSPGM that will solve the ODE using paths into the complex domain. Secondly, you must specify the path to be taken by the solution. This path is fixed by specifying the vertices of straight-line segments in the path. The path taken must be composed of straight-line segments. The first vertex is of course the starting point of the solution. A maximum of ten vertices may be specified. These vertices are to be placed into an array called POINTS, and the number of vertices used is stored in the variable

KPTS. The ATOMCC solution will follow the path thus specified and locate all the singularities near this path.

Since the ATOMCC solution will find all the singularities near the path specified by the user, certain problems may occur. First, the user may have by accident specified a path exactly midway between two singularities. In this event, there will be no information output from ATOMCC. The path must be slightly closer to one singularity than another; otherwise, ATOMCC cannot find the nearest singularity. Secondly, the user may have by accident specified a path that is too close to a singularity, or perhaps even a path that is directly pointed at a singularity. In this event, the ATOMCC solution will grind away and take very small steps. The information from ATOMCC beyond any such close encounter will be unreliable. In all cases, the user is well advised to change the path in the complex plane ever so slightly and make a second run to double check his results. In our experience, it is best to perform the complex integrations using double precision. Insert a COPTION DOUBLE,COMPLX card as the first card in your input. With just minimal experience, the user will find the use of ATOMCC in the complex plane to be fast and easy.

For good results, the first leg of the path into the complex domain should be directed straight up in the imaginary direction. Do not make the first leg of the path coincident with the real axis. This introduces subtraction errors into the complex solution.

When there are complex constants in the equations, the user is entirely responsible for seeing to it that those constants are properly specified with TYPE declarations in block 2, and the values are properly entered as CMLX(--,--) or DCMLX(--,--) in block 3. The reason for this requirement on the part of the user is because if the facts are otherwise, the user may then be required to use an editor to delete some possibly incorrect specification written by ATOMCC. It is better to be a bit short and correct than to be long and in error.

dragon.doc

TEXT
 Programming Insight: "DRAGON" Bruce R. Land.
 April page 137.

Screen #1
 (begin Dragon curve)
 CREATE CURVE (a FORGETable name)

 CARTESIAN OFF

 : RECURS SMUDGE ; IMMEDIATE (trick verb for recursion)

 VARIABLE ANGLE
 VARIABLE XCOORD
 VARIABLE YCOORD
 VARIABLE STEPSIZE

 : TURN (deltangle-- | turn sign*delta)
 ANGLE +! ;

 2 4 THRU

Screen #2
 : MOVE (-- | takes a step in present turtle direction)
 STEPSIZE @ DUP
 ANGLE @ COS * 10000 / (r* cos of theta) XCOORD @ +
 DUP (newX) XCOORD ! (update X)
 SWAP
 ANGLE @ SIN * 10000 / (r* sine theta) YCOORD @ +
 DUP (newY) YCOORD ! (update Y)
 DRAW.TO ;

Screen #3
 : DRAGON (sign level-- |)
 DUP (level) 0=
 IF (at bottom of recursion)
 DROP (level) DROP (sign) MOVE (by stepsize)
 ELSE
 OVER 45 * TURN (getsign and turn)
 1 (newsign)
 OVER 1- (level=level-1)
 RECURS DRAGON RECURS

 OVER -90 * TURN (getsign & turn)
 -1 (newsign) (edit to +1 for diff curve)
 OVER 1- (level=level-1)
 RECURS DRAGON RECURS
 DROP (input level) 45 * TURN (getsign and turn)
 THEN ;

Screen #4
 : DCURVE (level --|)
 (init pen position)
 PAGE 100 XCOORD ! 90 YCOORD ! 360 6 * ANGLE !
 WHITE PENPAT XCOORD @ YCOORD @ MOVE.TO

 PEN.NORMAL
 1 STEPSIZE !
 1 SWAP (level) DRAGON

 WHITE PENPAT 4 10 MOVE.TO PEN.NORMAL ;

(continued)

matinv.bas

TEXT

"The Inversion of Large Matrices" Thomas E. Phipps Jr.
April, page 181.

```

10 REM Real-entry matrix inversion (Pan-Reif method)
20 REM Program by Thomas E. Phipps, Jr.
30 REM Ref= Proc. 17th Annual ACM Sympos. on Theory
40 REM of Computing, Providence, RI, May 1985
50 REM Random-entry matrices, single precision
60 REM Quartic modification of Newton's iteration
70 INPUT "Run number = ";RN
80 REM N data is in line 640, D1 data in line 660
90 READ N, D1:RANDOMIZE RN:OPTION BASE 1
100 REM Can optionally insert DEFDBL A,B,E,X instruction here
110 REM A is the matrix to invert, B is the approximate inverse of A, E is the
error matrix for the inverse, and X is a temporary storage matrix as
explained in the BYTE article.
120 DIM A(N,N),B(N,N),E(N,N),X(N,N)
130 REM Generate random-entry A-matrix. For real data, replace 140 by an INPUT
statement and input normalized data (each element of A is divided by the
largest element, L). Delete line 150. B and D1 must be multiplied by L
later.
140 FOR I=1 TO N:FOR J=1 TO N:A(I,J)=RND(1):NEXT :NEXT
150 FOR I=1 TO N:FOR J=1 TO N:IF RND(1)<.5 THEN A(I,J)=-A(I,J)
160 NEXT :NEXT
170 CLS:TIME$="00" 'Start clock
180 REM Eval. t by Pan-Reif eq 8 in BYTE article
190 FOR I=1 TO N:R0=0:S0=0:FOR J=1 TO N
200 R0=R0+ABS(A(I,J)):S0=S0+ABS(A(J,I)):NEXT J
210 X(1,I)=R0:E(1,I)=S0:NEXT I:T1=0:T2=0
220 FOR I=1 TO N:IF X(1,I)>T1 THEN T1=X(1,I)
230 IF E(1,I)>T2 THEN T2=E(1,I)
240 NEXT I:T=1/(T1*T2)
250 REM Eval. initial B-matrix
260 FOR I=1 TO N:FOR J=1 TO N:B(I,J)=T*A(J,I):NEXT :NEXT :H=0
270 PRINT
"ITER."TAB(10);"E(1,1)"TAB(28);"E(1,N)"TAB(46);"B(1,1)"TAB(64);"B(1,N)"
280 REM Eval. error matrix E
290 FOR I=1 TO N:FOR K=1 TO N:Z=0:FOR J=1 TO N
300 Z=Z+B(I,J)*A(J,K):NEXT J:E(I,K)=-Z:NEXT K:NEXT I
310 FOR I=1 TO N:E(I,I)=1+E(I,I):NEXT I
320 BEEP:PRINT H;TAB(6);E(1,1);TAB(24);E(1,N);TAB(42);B(1,1);TAB(60);B(1,N)
330 IF H>50 THEN PRINT "STUCK!":BEEP:BEEP:GOTO 620
340 REM Test for escape from loop
350 FOR I=1 TO N:FOR J=1 TO N:IF ABS(E(I,J))>D1 THEN 370
360 NEXT :NEXT :GOTO 440
370 H=H+1 'Newton's iteration (quartic modification)
380 FOR I=1 TO N:X(1,I)=1+(1+(1+(1+E(I,I))*E(I,I))*E(I,I))*E(I,I):NEXT
390 FOR I=1 TO N:FOR J=1 TO N:E(I,J)=X(1,I)*E(I,J):NEXT :E(I,I)=1+E(I,I):NEXT
400 FOR I=1 TO N:FOR K=1 TO N:W=0:FOR J=1 TO N
410 W=W+E(I,J)*B(J,K):NEXT J:X(I,K)=W:NEXT K:NEXT I
420 FOR I=1 TO N:FOR J=1 TO N:B(I,J)=X(I,J):NEXT :NEXT
430 GOTO 280
440 REM output follows
450 T$=TIME$:BEEP:BEEP:BEEP:PRINT "Run number";RN,"Duration = ";T$
460 PRINT "No. iter.=";H,"Matrix dim.=";N,"Max. error=";D1
470 INPUT "Another iteration (y,n)";C$:IF C$="Y" OR C$="y" THEN
TIME$="00":GOTO 370
480 PRINT "The maximum error was calculated by multiplying A^-1*A."
490 PRINT "As a check on the true error in the approximated inverse matrix"
500 PRINT "the maximum error in A*A^-1 will now be determined."
510 ERASE X 'recycle the temporary storage matrix for errors in A*A^-1
520 DIM X(N,N)
530 FOR I=1 TO N:FOR J=1 TO N:FOR K=1 TO N
540 X(I,K)=X(I,K)+A(I,J)*B(J,K)
550 NEXT :NEXT :NEXT
560 REM You may print out the matrix and its inverse by placing a PRINT
command for A and B in the following loop.
570 FOR I=1 TO N:FOR J=1 TO N
580 TEST.VAL=ABS(X(I,J)):IF I=J THEN TEST.VAL=ABS(1-TEST.VAL)
590 IF TEST.VAL>MAX.ERROR THEN MAX.ERROR=TEST.VAL
600 NEXT :NEXT

```



```
610 PRINT "The max. error found this way is ";MAX.ERROR
620 END
630 REM Matrix dim. N =
640 DATA 10
650 REM Error criterion D1 =
660 DATA 3E-6
```


May

mainprog.bas

TEXT

Programming Insight: "Subroutine Overlays in GW-BASIC," Mike Carmichael.
May, page 151. Also download routine.sub.

```
1000 '    main program demo module
1005 '    MAINPROG
1010 '
1020 '
1030 COMMON
1040 DEFINT H,I,J,K,L
1060 LLOAD = 1
1070 FLNME$ = "routine.bin": D$ = "c:"
2000 '*****
2001 '    main driver
2002 '
2003 '
2020 GOSUB 12000 '    explanation
2030 GOSUB 8000 '    load 'routine.sub'
2040 GOSUB 30000 '    using ...
2050 GOSUB 31000 '    .... 'routine.sub'
2060 '*****
2070 '    normal program ending
2080 '
2100 STOP
8000 '*****
8001 '    load / save routine.sub
8002 '
8003 '
8010 GOSUB 10000 '    set lload switch
8020 ON ERROR GOTO 11000
8030 GOSUB 19900 '    position currpos
8040 IF LLOAD THEN BLOAD FL$, CURRPOS ELSE BSAVE FL$,
    CURRPOS, 1000
8050 IF LLOAD THEN GOSUB 8100
8060 RETURN
8100 '*****
8101 '    adjust binary line
8102 '    pointers when loading binary
8103 '    subroutines
8120 GOSUB 9800 '    calculate offset
8130 IF OFFSET = 0 THEN RETURN
8140 WHILE PEEK(CURRPOS+1) > 0
8150     VARRBLE = NADDRESS + OFFSET
8160     GOSUB 9900 '    split varrble
8170     GOSUB 8600 '    store varrble
8180     CURRPOS = VARRBLE
8190     GOSUB 8700 '    get next address
8200 WEND
8210 RETURN
8600 '*****
8601 '    store next program line address
8602 '    at currpos and increment
8603 '    currpos
8640 POKE CURRPOS, LOW
8650 POKE CURRPOS+1, HIGH
8670 RETURN
8700 '*****
8701 '    get next line address from
8702 '    memory
8703 '
8710 LOW = PEEK(CURRPOS)
8720 HIGH = PEEK(CURRPOS+1)
8730 NADDRESS = HIGH * 256 + LOW
8740 RETURN
```

(continued)

```

9800 '*****
9801 '   calculate offset -- length
9802 '   of rem statement ( 8 bytes )
9803 '   in the first line of
9804 '   subroutine
9810 GOSUB 8700 ' next address
9820 OFFSET = CURRPOS + 8 - NADDRESS
9830 RETURN
9900 '*****
9901 '   split 'varrrble' address or
9902 '   number for storage
9903 '   'variables expected' varrrble
9904 '   'variables returned' high, low
9910 HIGH = INT(VARRBLE / 256)
9920 LOW = VARRBLE - (256 * HIGH)
9930 RETURN
10000 '*****
10001 '   BLOAD 'routine.sub'
10002 '   -- file found --
10003 '   proceed to load it
10004 '
10010 ON ERROR GOTO 10500 'error in open
10020 FL$ = D$ + FLNME$
10030 OPEN "i", 1, FL$
10040 CLOSE
10050 RETURN
10100 '*****
10101 '   BSAVE 'routine.sub'
10102 '   -- file not found --
10103 '   proceed to save it
10104 '
10110 LLOAD = 0
10120 RETURN
10500 '   error in opening 'routine.sub'
10510 IF ERR = 53 THEN RESUME 10100
10520 PRINT "*** error in opening "; FL$; " ***";: ' fatal
10530 STOP
11000 '*****
11001 '   error in BSAVE or BLOAD process
11002 '
11003 '
11020 PRINT "*** error in processing "; F$; " ***";: ' fatal
11030 STOP
11050 RETURN
12000 '*****
12001 '   explanation of 'mainprog'
12002 '   and 'routine.sub'
12003 '
12010 CLS
12020 LOCATE 1,1
12030 COLOR 0,7
12040 '
12050 LOCATE 1,57
12060 PRINT "Date "; DATE$; " "; TIME$;
12070 LOCATE 2,15
12080 PRINT "A sample of a BASIC program calling a BASIC
  subroutine"
12090 COLOR 7,0
12110 LOCATE 5,1
12120 PRINT "'mainprog' has been written to demonstrate a
  method we have devised
12130 PRINT "to facilitate calling BASIC subroutines using
  Microsoft BASIC.
12140 PRINT ""
12150 PRINT "the modules starting at line 8000 either load
  or save a subroutine
12160 PRINT "consisting of two modules -- starting at lines
  30000 and 31000 --
12170 PRINT "in binary, and then update consecutive line
  pointers to agree with
12180 PRINT "the correct positions in memory "
12190 PRINT ""
12200 PRINT "subroutines at lines 30000 and 31000 are
  executed after they have
12210 PRINT "been loaded into memory from their binary file
  'routine.bin'

```


BYTE LISTINGS SUPPLEMENT 319

```

20065 '*****
*****
*****fill*
*****
*****'
20070 '*****
*****fill*
*****
*****'
20075 '*****
*****
*****
*****
*****'
29985 ' set variable at end of main program when saving
routine on disk
29990 F$ = "*"
29995 RETURN

```

huffman.bix

TEXT

Programming Project: "Data Compression with Huffman Coding," Jonathan Amsterdam.
May, page 98. Also download hufread.me

```

=====
Editor's note: Remember to break the following modules into their own files
before attempting to compile them. Delete "=====" and comments inside them
=====

```

```

Start BitStream.DEF
=====

```

```

DEFINITION MODULE BitStream;

```

```

(* Used for bit-oriented I/O. Minimal facilities. *)

```

```

EXPORT QUALIFIED connect, disconnect, EOS, read, write, bitStream,
readChar, writeChar, readCard, writeCard;

```

```

TYPE bitStream;

```

```

PROCEDURE connect(fileName:ARRAY OF CHAR; read:BOOLEAN):bitStream;
(* Associates a file with a bitStream. A given stream can be
read from or written to, but not both. On a Mac, this procedure
uses the default drive. *)

```

```

PROCEDURE disconnect(bs:bitStream);
(* Disconnects stream from file. *)

```

```

PROCEDURE EOS(bs:bitStream):BOOLEAN;
(* TRUE at end of stream; for read streams only! *)

```

```

PROCEDURE read(bs:bitStream):BOOLEAN;
(* Reads a bit from the stream. TRUE = 1, FALSE = 0 *)

```

```

PROCEDURE write(bs:bitStream; b:BOOLEAN);
(* Writes a bit to the stream. *)

```

```

PROCEDURE readChar(bs:bitStream):CHAR;
(* Reads eight consecutive bits and translates them into a CHAR. This is
somewhat implementation-dependent. *)

```

```

PROCEDURE writeChar(bs:bitStream; c:CHAR);
(* Writes the character as eight consecutive bits. This is somewhat
implementation-dependent. *)

```

```

PROCEDURE readCard(bs:bitStream):CARDINAL;
(* Reads 16 consecutive bits and translates them into a CARDINAL. *)

```

```

PROCEDURE writeCard(bs:bitStream; c:CARDINAL);
(* Writes the cardinal as 16 consecutive bits *)

```


END BitStream.

=====

Start BitStream.MOD

=====

IMPLEMENTATION MODULE BitStream;

(* Note: because Streams.WriteWord and Streams.ReadWord don't appear to work in MacModula-2, I do I/O with the character operations. A character code occupies bits 8-15 of a word. *)

FROM Streams IMPORT STREAM, StreamType, Connect, Disconnect, ReadChar, WriteChar;

IMPORT Streams;

FROM Storage IMPORT ALLOCATE, DEALLOCATE;

FROM MyTerminal IMPORT fatal;

CONST maxBit = 15; (* highest numbered bit in a BITSET *)
lowCharBit = 8; (* first bit of a character *)

TYPE wordRange = [0..maxBit];
charRange = [lowCharBit..maxBit];
bitStream = POINTER TO bsRec;
bsRec = RECORD
 stream:STREAM;
 read:BOOLEAN;
 curWord:BITSET;
 curBit:charRange;
END;

PROCEDURE connect(fileName:ARRAY OF CHAR; read:BOOLEAN):bitStream;

VAR bs:bitStream;

 nullVol:ARRAY[0..0] OF CHAR;

 done:BOOLEAN;

BEGIN

 nullVol[0] := 0C;

 NEW(bs);

 bs^.read := read;

 IF read THEN

 Connect(bs^.stream, streamread, (* open stream for reading *)
 fileName, nullVol, 1, (* use drive #1 *)
 FALSE, (* don't create if nonexistent *)
 done);

 IF NOT done THEN

 fatal('cannot open file');

 END;

 bs^.curBit := maxBit;

 ELSE

 Connect(bs^.stream, streamwrite, fileName, nullVol, 1, TRUE, done);

 IF NOT done THEN

 fatal('cannot open file');

 END;

 bs^.curBit := lowCharBit;

 bs^.curWord := {};

 END;

 RETURN bs;

END connect;

PROCEDURE disconnect(bs:bitStream);

BEGIN

 WITH bs^ DO

 IF (NOT read) AND (curBit <> lowCharBit) THEN (* flush the last word *)
 WriteChar(stream, CHAR(curWord));

 END;

 Disconnect(stream);

 DISPOSE(bs);

END;

END disconnect;

(continued)

```

PROCEDURE EOS(bs:bitStream):BOOLEAN;
BEGIN
  WITH bs^ DO
    IF read THEN
      RETURN (curBit = maxBit) AND Streams.EOS(stream);
    ELSE
      fatal('EOS called on write bit stream');
    END;
  END;
END EOS;

PROCEDURE read(bs:bitStream):BOOLEAN;
(* Init: curBit := maxBit. curBit = "all bits to curBit have been read" *)
VAR c:CHAR;
BEGIN
  IF NOT bs^.read THEN
    fatal('attempt to read a write bit stream');
  ELSE WITH bs^ DO
    IF curBit = maxBit THEN
      IF NOT Streams.EOS(stream) THEN
        ReadChar(stream, c);
        curWord := BITSET(c);
        curBit := lowCharBit;
      END;
    ELSE
      INC(curBit);
    END;
    RETURN curBit IN curWord;
  END; END;
END read;

PROCEDURE write(bs:bitStream; b:BOOLEAN);
(* init: curBit := lowCharBit, curWord := {}.
  curBit = "bit curBit is next to be written" *)
BEGIN
  WITH bs^ DO
    IF read THEN
      fatal('attempt to write a read bit stream');
    END;
    IF b THEN
      INCL(curWord, curBit);
    END;
    IF curBit = maxBit THEN
      WriteChar(stream, CHAR(curWord));
      curWord := {};
      curBit := lowCharBit;
    ELSE
      INC(curBit);
    END;
  END;
END write;

PROCEDURE readChar(bs:bitStream):CHAR;
(* Read 8 bits and make them into a character. In MacModula-2,
  a CHAR variable is a word with bits 8-15 containing the ASCII code. *)
VAR i:charRange;
  char:BITSET;
BEGIN
  char := {};
  FOR i := lowCharBit TO maxBit DO
    IF read(bs) THEN
      INCL(char, i);
    END;
  END;
  RETURN CHAR(char);
END readChar;

PROCEDURE writeChar(bs:bitStream; c:CHAR);
(* see readChar for implementation details *)
VAR i:charRange;

```



```

BEGIN
  FOR i := lowCharBit TO maxBit DO
    write(bs, i IN BITSET(c));
  END;
END writeChar;

PROCEDURE readCard(bs:bitStream):CARDINAL;
VAR i:wordRange;
    card:BITSET;
BEGIN
  FOR i := 0 TO maxBit DO
    IF read(bs) THEN
      INCL(card, i);
    ELSE
      EXCL(card, i);
    END;
  END;
  RETURN CARDINAL(card);
END readCard;

PROCEDURE writeCard(bs:bitStream; c:CARDINAL);
VAR i:wordRange;
BEGIN
  FOR i := 0 TO maxBit DO
    write(bs, i IN BITSET(c));
  END;
END writeCard;

BEGIN
END BitStream.

=====
Start CharStream.DEF
=====

DEFINITION MODULE CharStream;

EXPORT QUALIFIED charStream, connect, disconnect, read, write, EOS;

TYPE
  charStream;

PROCEDURE connect(fileName:ARRAY OF CHAR; read:BOOLEAN):charStream;
PROCEDURE disconnect(cs:charStream);
PROCEDURE read(cs:charStream):CHAR;
PROCEDURE write(cs:charStream; c:CHAR);
PROCEDURE EOS(cs:charStream):BOOLEAN;
END CharStream.

=====
Start CharStream.MOD
=====

IMPLEMENTATION MODULE CharStream;

(* This module supports character I/O from files. Its facilities are minimal.
   I wrote it using MacModula-2's Streams module; it should be easy to
   duplicate its behavior with whatever file system you have. *)

FROM Streams IMPORT STREAM, StreamType, Connect, Disconnect,
  ReadChar, WriteChar;
IMPORT Streams;
FROM MyTerminal IMPORT fatal;
FROM Storage IMPORT ALLOCATE, DEALLOCATE;

TYPE
  charStream = POINTER TO STREAM;

```

(continued)

```
PROCEDURE connect(fileName:ARRAY OF CHAR; read:BOOLEAN):charStream;
```

```
VAR cs:charStream;
```

```
    nullVol: ARRAY[0..0] OF CHAR;
```

```
    done:BOOLEAN;
```

```
BEGIN
```

```
    nullVol[0] := 0C;
```

```
    NEW(cs);
```

```
    IF read THEN
```

```
        Connect(cs^, streamread,      (* open stream for reading *)
                fileName, nullVol, 1, (* use drive #1 *)
                FALSE,                (* don't create if nonexistent *)
                done);
```

```
        IF NOT done THEN
```

```
            fatal('cannot open file');
```

```
        END;
```

```
    ELSE
```

```
        Connect(cs^, streamwrite, fileName, nullVol, 1, TRUE, done);
```

```
        IF NOT done THEN
```

```
            fatal('cannot open file');
```

```
        END;
```

```
    END;
```

```
    RETURN cs;
```

```
END connect;
```

```
PROCEDURE disconnect(cs:charStream);
```

```
BEGIN
```

```
    Disconnect(cs^);
```

```
    DISPOSE(cs);
```

```
END disconnect;
```

```
PROCEDURE read(cs:charStream):CHAR;
```

```
VAR c:CHAR;
```

```
BEGIN
```

```
    ReadChar(cs^, c);
```

```
    RETURN c;
```

```
END read;
```

```
PROCEDURE write(cs:charStream; c:CHAR);
```

```
BEGIN
```

```
    WriteChar(cs^, c);
```

```
END write;
```

```
PROCEDURE EOS(cs:charStream):BOOLEAN;
```

```
BEGIN
```

```
    RETURN Streams.EOS(cs^);
```

```
END EOS;
```

```
BEGIN
```

```
END CharStream.
```

```
=====
```

```
Start Compress.MOD
```

```
=====
```

```
MODULE Compress;
```

```
(* File compression algorithm using Huffman coding.
```

```
Based on "Data Compression with Huffman Coding," BYTE March 1986.
```

```
Copyright 1986 by Jonathan Amsterdam. All Rights Reserved.
```

```
*)
```

```
FROM CharStream IMPORT charStream;
```

```
IMPORT CharStream;
```

```
FROM BitStream IMPORT bitStream;
```

```
IMPORT BitStream;
```

```
FROM MyTerminal IMPORT ClearScreen, pause, WriteCard, WriteLn, WriteString,
```

```
WriteLnString, Write;
```

```
FROM InOut IMPORT ReadString;
```

```
FROM Huffman IMPORT huffTree, huffman, readCode, writeCode, readTree,
```

```
writeTree, codeSize;
```

```
FROM StringStuff IMPORT stringLen, stringCopy;
```

```
FROM RealInOut IMPORT FWriteReal;
```

```
(* FWriteReal writes real numbers in decimal format. If your implementation
```

```
doesn't have it, substitute WriteReal. *)
```



```

CONST stringlen = 60;

VAR frequency:ARRAY CHAR OF CARDINAL;
    fileSize:CARDINAL;
    inFileName, outFileName: ARRAY[0..stringlen] OF CHAR;
    hTree:huffTree;

PROCEDURE doFreq;
(* Obtain frequency count from file *)
VAR cs:charStream;
BEGIN
    cs := CharStream.connect(inFileName, TRUE);          (* read file *)
    freqCount(cs);
    CharStream.disconnect(cs);
END doFreq;

PROCEDURE freqCount(cs:charStream);
VAR c:CHAR;
BEGIN
    FOR c := 0C TO CHR(HIGH(frequency)) DO
        frequency[c] := 0;
    END;
    c := CharStream.read(cs);
    WHILE NOT CharStream.EOS(cs) DO
        INC(frequency[c]);
        INC(fileSize);
        c := CharStream.read(cs);
    END;
END freqCount;

PROCEDURE doOutput;
(* Output encoded file *)
VAR inStream:charStream;
    outStream:bitStream;
    c:CHAR;
BEGIN
    inStream := CharStream.connect(inFileName, TRUE);
    outStream := BitStream.connect(outFileName, FALSE);
    BitStream.writeCard(outStream, fileSize);
    writeTree(outStream, hTree);
    c := CharStream.read(inStream);
    WHILE NOT CharStream.EOS(inStream) DO
        writeCode(outStream, hTree, c);
        c := CharStream.read(inStream);
    END;
    CharStream.disconnect(inStream);
    BitStream.disconnect(outStream);
END doOutput;

PROCEDURE computeStats;
(* Compute statistics on how much space was saved *)
VAR c:CHAR;
    origBits, compBits, nChars:CARDINAL;
    savings:REAL;
BEGIN
    origBits := fileSize * 8;
    compBits := 0;

    nChars := 0;
    FOR c := 0C TO CHR(HIGH(frequency)) DO
        IF frequency[c] <> 0 THEN
            INC(nChars);
            compBits := compBits + codeSize(hTree, c) * frequency[c];
        END;
    END;
    WriteString("number of different characters: ");
    WriteCard(nChars, 0); WriteLn;
    WriteString("original file size (bits): ");
    WriteCard(origBits, 0); WriteLn;

```

(continued)

```

WriteString("compressed f. size (bits): ");
WriteCard(compBits, 0); WriteLn;
WriteString("percent savings: ");
savings := 1.0 - (FLOAT(compBits) / FLOAT(origBits));
FWriteReal(savings * 100.0, 5); WriteLn;
WriteString("compressed size, including bookkeeping: ");
(* add 16 bits for character, count, 10n-1 bits for tree *)
INC(compBits, 16 + 10*nChars - 1);
WriteCard(compBits, 0); WriteLn;
WriteString("true percent savings: ");
savings := 1.0 - (FLOAT(compBits) / FLOAT(origBits));
FWriteReal(savings * 100.0, 5); WriteLn;
END computeStats;

PROCEDURE doOutfileName;
(* Make the name of the output file by appending ".P" to the input file's
   name *)
VAR len: CARDINAL;
BEGIN
    len := stringLen(inFileName);
    stringCopy(outFileName, inFileName);
    outFileName[len] := '.';
    outFileName[len+1] := 'P';
    outFileName[len+2] := 0C;
END doOutfileName;

BEGIN
    ClearScreen;
    WriteLnString("File Compression using Huffman Coding");
    WriteString("Input file: ");
    ReadString(inFileName);

    doOutfileName;
    doFreq;
    hTree := huffman(frequency);
    doOutput;
    computeStats;
    pause('done--');
END Compress.

=====
Start Huffman.DEF
=====

DEFINITION MODULE Huffman;

(* Implements the Huffman coding scheme and procedures for manipulating
   the code tree. *)

FROM BitStream IMPORT bitStream;

EXPORT QUALIFIED huffTree, huffman, writeCode, readCode, writeTree, readTree,
    codeSize;

TYPE huffTree;

PROCEDURE huffman(VAR frequency: ARRAY OF CARDINAL): huffTree;
(* construct a Huffman coding tree from the given character frequencies *)

PROCEDURE writeCode(bs: bitStream; ht: huffTree; c: CHAR);
(* Write the code for c onto bs, using ht. *)

PROCEDURE readCode(bs: bitStream; ht: huffTree): CHAR;
(* Read bits from bs until a full code is read; return the character *)

PROCEDURE writeTree(bs: bitStream; ht: huffTree);
(* Write the tree onto the stream *)

PROCEDURE readTree(bs: bitStream): huffTree;
(* Read a huffTree from the stream *)

PROCEDURE codeSize(ht: huffTree; c: CHAR): CARDINAL;
(* returns the length of the code for c *)

END Huffman.

```



```
=====
Start Huffman.MOD
=====
```

```
IMPLEMENTATION MODULE Huffman;
```

```
(* Huffman coding algorithm, as described in "Data Compression With Huffman
Coding," BYTE, March 1986.
Copyright Jonathan Amsterdam 1986, All Rights Reserved. *)
```

```
FROM Storage IMPORT ALLOCATE, DEALLOCATE;
FROM BitStream IMPORT bitStream;
IMPORT BitStream;
FROM MyTerminal IMPORT WriteString, WriteCard, WriteLn, fatal;
CONST maxChars = 256;
```

```
TYPE
  node = POINTER TO nodeRec;
  nodeRec = RECORD
    char:CHAR;
    freq:CARDINAL;
    child:ARRAY BOOLEAN OF node;
    parent:node;      (* used for encoding *)
  END;

  huffTree = POINTER TO htRec;
  htRec = RECORD
    tree:node;      (* the tree itself *)
    leaf:ARRAY CHAR OF node; (* index by character, for encoding
*)
  END;
```

```
VAR tree:ARRAY[1..maxChars] OF node;  (* temporary list of trees *)
    nTrees:CARDINAL;
```

```
    (** constructing the tree **)
```

```
PROCEDURE huffman(VAR frequency:ARRAY OF CARDINAL):huffTree;
VAR ht:huffTree;
BEGIN
  ht := initHuffTree(frequency);
  initTrees(ht^.leaf);
  WHILE nTrees > 1 DO
    insert(combineNodes(removeSmallest(), removeSmallest()));
  END;
  ht^.tree := tree[1];
  RETURN ht;
END huffman;
```

```
PROCEDURE initHuffTree(VAR freq:ARRAY OF CARDINAL):huffTree;
VAR i:CARDINAL;

  ht:huffTree;
BEGIN
  ht := newHuffTree();
  FOR i := 0 TO HIGH(freq) DO
    IF freq[i] <> 0 THEN
      ht^.leaf[CHR(i)] := newNode(CHR(i), freq[i], NIL, NIL);
    END;
  END;
  RETURN ht;
END initHuffTree;
```

```
PROCEDURE initTrees(VAR leaf:ARRAY OF node);
VAR i:CARDINAL;
BEGIN
  nTrees := 0;
  FOR i := 0 TO HIGH(leaf) DO
    IF leaf[i] <> NIL THEN
      insert(leaf[i]);
    END;
  END;
END;
END initTrees;
```

(continued)

```

PROCEDURE removeSmallest():node;
VAR i, smallest:CARDINAL;
    smallestNode:node;
BEGIN
    smallest := 1;
    FOR i := 2 TO nTrees DO
        IF tree[i]^freq < tree[smallest]^freq THEN
            smallest := i;
        END;
    END;
    smallestNode := tree[smallest];
    tree[smallest] := tree[nTrees];
    DEC(nTrees);
    RETURN smallestNode;
END removeSmallest;

PROCEDURE insert(n:node);
BEGIN
    INC(nTrees);
    tree[nTrees] := n;
END insert;

    (** code I/O **)

PROCEDURE writeCode(bs:bitStream; ht:huffTree; c:CHAR);
(* Write the code for c onto bs, using ht. By using recursion, we can
   avoid explicitly retracing the path from the root to the leaf. *)
PROCEDURE wrCode(n:node);
BEGIN
    IF n^.parent <> NIL THEN
        wrCode(n^.parent);
        BitStream.write(bs, n = n^.parent^.child[TRUE]);
    END;
END wrCode;

BEGIN
    IF ht^.leaf[c] = NIL THEN
        WriteString("no code for "); WriteCard(CARDINAL(c), 0); WriteLn;
        fatal('dying');
    END;
    wrCode(ht^.leaf[c]);
END writeCode;

PROCEDURE readCode(bs:bitStream; ht:huffTree):CHAR;
(* Read bits from bs until a full code is read; return the character *)
PROCEDURE rdCode(n:node):CHAR;
BEGIN
    IF leaf(n) THEN
        RETURN n^.char;
    ELSE
        RETURN rdCode(n^.child[BitStream.read(bs)]);
    END;
END rdCode;

BEGIN
    RETURN rdCode(ht^.tree);
END readCode;

PROCEDURE writeTree(bs:bitStream; ht:huffTree);
(* Write the tree onto the stream. It is encoded as follows:
   A 1 bit indicates an internal node.
   A 0 bit indicates a leaf; the next 8 bits are the character code.
   The tree is traversed by preorder traversal: first the root, then
   the left (FALSE) subtree, then the right (TRUE). *)
PROCEDURE wrTree(n:node);
BEGIN
    IF leaf(n) THEN
        BitStream.write(bs, FALSE);
        BitStream.writeChar(bs, n^.char);
    END;
END wrTree;

```



```

        ELSE
            BitStream.write(bs, TRUE);
            wrTree(n^.child[FALSE]);
            wrTree(n^.child[TRUE]);
        END;
    END wrTree;

BEGIN
    wrTree(ht^.tree);
END writeTree;

PROCEDURE readTree(bs:bitStream):huffTree;
(* Read a huffTree from the stream. See writeTree for the encoding used.
   Frequency information is NOT preserved. *)
VAR ht:huffTree;

    PROCEDURE rdTree():node;
    VAR false, true, n:node;
    BEGIN
        IF BitStream.read(bs) THEN (* an internal node *)
            false := rdTree();
            true := rdTree();
            n := newNode(0C, 0, false, true);
            false^.parent := n;
            true^.parent := n;
            RETURN n;
        ELSE (* a leaf *)
            n := newNode(BitStream.readChar(bs), 0, NIL, NIL);
            ht^.leaf[n^.char] := n;
            RETURN n;
        END;
    END rdTree;

    BEGIN
        ht := newHuffTree();
        ht^.tree := rdTree();
        RETURN ht;
    END readTree;

    (** huffTree allocation **)

PROCEDURE newHuffTree():huffTree;
VAR c:CHAR;
    ht:huffTree;
    BEGIN
        NEW(ht);
        FOR c := 0C TO CHR(HIGH(ht^.leaf)) DO
            ht^.leaf[c] := NIL;
        END;
        RETURN ht;
    END newHuffTree;

    (** node stuff **)

PROCEDURE combineNodes(n1, n2:node):node;
(* used to combine nodes when constructing the coding tree *)
VAR n:node;
    BEGIN
        n := newNode(0C, n1^.freq + n2^.freq, n1, n2);
        n1^.parent := n;
        n2^.parent := n;
        RETURN n;
    END combineNodes;

PROCEDURE newNode(c:CHAR; f:CARDINAL; false, true:node):node;
VAR n:node;
    BEGIN
        NEW(n);
        WITH n^ DO
            char := c;
            freq := f;
            child[FALSE] := false;

```

(continued)

May

```
        child[TRUE] := true;
        parent := NIL;
    END;
    RETURN n;
END newNode;

PROCEDURE freeNode(n:node);
(* In the current implementation, this is never used *)
BEGIN
    IF n <> NIL THEN
        freeNode(n^.child[FALSE]);
        freeNode(n^.child[TRUE]);
        DISPOSE(n);
    END;
END freeNode;

PROCEDURE leaf(n:node):BOOLEAN;
BEGIN
    IF n = NIL THEN
        fatal('leaf: n NIL');
    END;
    RETURN n^.child[FALSE] = NIL;
END leaf;

PROCEDURE codeSize(ht:huffTree; c:CHAR):CARDINAL;
(* returns the length of the code for c *)
VAR i:CARDINAL;
    n:node;
BEGIN
    i := 0;
    n := ht^.leaf[c];
    WHILE n <> NIL DO
        INC(i);
        n := n^.parent;
    END;
    RETURN i-1;
END codeSize;

BEGIN
    END Huffman.

=====
Start MyTerminal.DEF
=====

DEFINITION MODULE MyTerminal;

(* Some small but useful additions to the Terminal module. *)

EXPORT QUALIFIED WriteString, WriteLn, Write, Read, ClearScreen, Beep,
    WriteLnString, WriteInt, WriteCard, pause, fatal;

PROCEDURE WriteString(s:ARRAY OF CHAR);
PROCEDURE WriteLn;
PROCEDURE Write(c:CHAR);
PROCEDURE Read(VAR c:CHAR);
PROCEDURE ClearScreen;
PROCEDURE Beep;

PROCEDURE WriteLnString(s:ARRAY OF CHAR);
PROCEDURE WriteInt(i:INTEGER; spaces:CARDINAL);
PROCEDURE WriteCard(c, spaces:CARDINAL);

PROCEDURE pause(msg:ARRAY OF CHAR);
(* Prevents the screen from blanking and returning to the Finder until the
   user hits a key. msg is typed out. *)

PROCEDURE fatal(msg:ARRAY OF CHAR);
(* Prints the message, does a pause, and HALTs. *)

END MyTerminal.
```



```

=====
Start MyTerminal.MOD
=====

IMPLEMENTATION MODULE MyTerminal;

(* Some small but useful additions to the Terminal module. *)

IMPORT Terminal;

VAR powerOfTen: ARRAY[0..4] OF CARDINAL;

PROCEDURE WriteLnString(s:ARRAY OF CHAR);
BEGIN
    Terminal.WriteString(s);
    Terminal.WriteLine;
END WriteLnString;

PROCEDURE WriteInt(i:INTEGER; spaces:CARDINAL);
BEGIN
    IF i < 0 THEN
        writeNum(CARDINAL(-i), spaces-1, TRUE);
    ELSE
        writeNum(CARDINAL(i), spaces, FALSE);
    END;
END WriteInt;

PROCEDURE WriteCard(c, spaces:CARDINAL);
BEGIN
    writeNum(c, spaces, FALSE);
END WriteCard;

PROCEDURE writeNum(c, spaces:CARDINAL; neg:BOOLEAN);
VAR p:CARDINAL;
    i:INTEGER;
BEGIN
    p := places(c);
    FOR i := 1 TO INTEGER(spaces) - INTEGER(p) DO
        Terminal.Write(' ');
    END;
    IF neg THEN
        Terminal.Write('-');
    END;
    FOR i := p-1 TO 0 BY -1 DO
        Terminal.Write(CHR((c DIV powerOfTen[i]) + ORD('0')));
        c := c MOD powerOfTen[i];
    END;
END writeNum;

PROCEDURE places(c:CARDINAL):CARDINAL;
(* Returns the number of places c takes to print; i.e. trunc(1+log10(c)). *)
VAR i:CARDINAL;
BEGIN
    FOR i := 4 TO 0 BY -1 DO
        IF (c DIV powerOfTen[i]) > 0 THEN
            RETURN i+1;
        END;
    END;
    RETURN 1;
END places;

PROCEDURE pause(msg:ARRAY OF CHAR);

(* Prevents the screen from blanking and returning to the Finder until the
   user hits a key. msg is typed out. *)
VAR ch:CHAR;
BEGIN
    Terminal.WriteString(msg);
    Terminal.Read(ch);
END pause;

PROCEDURE fatal(msg:ARRAY OF CHAR);

```

(continued)

May

```
BEGIN
  WriteLnString(msg);
  pause('Hit any key to die--');
  HALT;
END fatal;
```

(*** Copies of Terminal procedures ***)

```
PROCEDURE WriteString(s:ARRAY OF CHAR);
```

```
BEGIN
  Terminal.WriteString(s);
END WriteString;
```

```
PROCEDURE WriteLn;
```

```
BEGIN
  Terminal.WriteLn;
END WriteLn;
```

```
PROCEDURE Write(c:CHAR);
```

```
BEGIN
  Terminal.Write(c);
END Write;
```

```
PROCEDURE Read(VAR c:CHAR);
```

```
BEGIN
  Terminal.Read(c);
END Read;
```

```
PROCEDURE ClearScreen;
```

```
BEGIN
  Terminal.ClearScreen;
END ClearScreen;
```

```
PROCEDURE Beep;
```

```
BEGIN
  Terminal.Beep;
END Beep;
```

```
BEGIN
```

```
  powerOfTen[0] := 1;
  powerOfTen[1] := 10;
  powerOfTen[2] := 100;
  powerOfTen[3] := 1000;
  powerOfTen[4] := 10000;
```

```
END MyTerminal.
```

```
=====
Start StringStuff.DEF
=====
```

```
DEFINITION MODULE StringStuff;
```

```
EXPORT QUALIFIED stringCap, charCap, stringLen, stringCopy, stringEqual;
```

```
PROCEDURE charCap(ch:CHAR):CHAR;
```

```
PROCEDURE stringCap(VAR s:ARRAY OF CHAR);
```

```
PROCEDURE stringLen(VAR s:ARRAY OF CHAR):CARDINAL;
```

```
PROCEDURE stringCopy(VAR s1:ARRAY OF CHAR; s2:ARRAY OF CHAR);
```

```
PROCEDURE stringEqual(s1, s2:ARRAY OF CHAR):BOOLEAN;
```

```
END StringStuff.
```

```
=====
Start StringStuff.MOD
=====
```

```
IMPLEMENTATION MODULE StringStuff;
```



```

PROCEDURE charCap(ch:CHAR):CHAR;
BEGIN
  IF (ch >= 'a') AND (ch <= 'z') THEN
    RETURN CAP(ch);
  ELSE
    RETURN ch;
  END;
END charCap;

PROCEDURE stringCap(VAR s:ARRAY OF CHAR);
VAR i:CARDINAL;
BEGIN
  FOR i := 0 TO stringLen(s) DO
    s[i] := charCap(s[i]);
  END;
END stringCap;

PROCEDURE stringLen(VAR s:ARRAY OF CHAR):CARDINAL;
VAR i:CARDINAL;
BEGIN
  FOR i := 0 TO HIGH(s) DO
    IF s[i] = 0C THEN
      RETURN i;
    END;
  END;
  RETURN HIGH(s)+1;
END stringLen;

PROCEDURE stringCopy(VAR s1:ARRAY OF CHAR; s2:ARRAY OF CHAR);
VAR i:CARDINAL;
BEGIN
  i := 0;
  LOOP
    IF i > HIGH(s1) THEN
      EXIT;
    ELSIF i > HIGH(s2) THEN
      s1[i] := 0C;
      EXIT;
    ELSE
      s1[i] := s2[i];
    END;
    INC(i);
  END;
END stringCopy;

PROCEDURE stringEqual(s1, s2:ARRAY OF CHAR):BOOLEAN;
VAR i:CARDINAL;
BEGIN
  FOR i := 0 TO HIGH(s1) DO
    IF i > HIGH(s2) THEN
      RETURN s1[i] = 0C;
    ELSIF s1[i] <> s2[i] THEN
      RETURN FALSE;
    ELSIF s1[i] = 0C THEN
      RETURN TRUE;
    END;
  END;
  RETURN TRUE;
END stringEqual;

BEGIN
END StringStuff.

=====
Start Uncompress.MOD
=====

MODULE Uncompress;

(* Takes files encoded by Compress and restores them to their original
state.
Copyright 1986 by Jonathan Amsterdam. All Rights Reserved. *)

```

(continued)

```

FROM CharStream IMPORT charStream;
IMPORT CharStream;
FROM BitStream IMPORT bitStream;
IMPORT BitStream;
FROM MyTerminal IMPORT ClearScreen, pause, WriteCard, WriteLn, WriteString,
    WriteLnString, Write;
FROM InOut IMPORT ReadString;
FROM Huffman IMPORT huffTree, huffman, readCode, readTree;
FROM StringStuff IMPORT stringLen, stringCopy;

```

```
CONST stringlen = 60;
```

```
VAR inFileName, outFileName: ARRAY[0..stringlen] OF CHAR;
```

```

PROCEDURE doUncompress;
VAR inStream:bitStream;
    outStream:charStream;
    fileSize, i:CARDINAL; (* number of characters in file *)
    hTree:huffTree;

```

```

BEGIN
    inStream := BitStream.connect(inFileName, TRUE);
    outStream := CharStream.connect(outFileName, FALSE);
    fileSize := BitStream.readCard(inStream);
    hTree := readTree(inStream);
    FOR i := 1 TO fileSize DO
        CharStream.write(outStream, readCode(inStream, hTree));
    END;
    CharStream.disconnect(outStream);
    BitStream.disconnect(inStream);
END doUncompress;

```

```

PROCEDURE doFileNames;
VAR len:CARDINAL;
BEGIN
    len := stringLen(inFileName);
    stringCopy(outFileName, inFileName);
    inFileName[len] := '.';
    inFileName[len+1] := 'P';
    inFileName[len+2] := 0C;
    outFileName[len] := '.';
    outFileName[len+1] := 'U';
    outFileName[len+2] := 0C;
END doFileNames;

```

```

BEGIN
    ClearScreen;
    WriteLnString("Uncompression program");
    WriteString('Input file (omit ".P") : ');
    ReadString(inFileName);
    doFileNames;
    doUncompress;
    pause('done--');
END Uncompress.

```

hufread.me

TEXT

Programming Project: "Data Compression with Huffman Coding," Jonathan Amsterdam.
May, page 98. Also download huffman.bix.

This is the Modula-2 source code accompanying Jonathan Amsterdam's article "Data Compression with Huffman Coding" in May BYTE 1986. This code was developed under MacModula-2 for the Mac but should be easy to convert to other Modula-2 compilers. Because the Mac allows character names longer than eight, the modules have been concatenated into one file with comments separating the different modules. You should break these modules out into their own file before attempting to compile them.

optidb.c

TEXT

"The Application Interface of Optical Drives," by Jeffrey R. Dulude.
May, page 193.

A Write Once Corporate Personnel Database

Purpose

The purpose of this database is to provide an example of how a write once application could be developed. Code is provided only to the point of being followable for concept explanation and development. Some of the functions are available from the Optotech "C" interface library which runs on the Optotech 200 Megabyte/side write once optical disk.

Concepts

Several concepts are used here:

- space is cheap -- there's no need to throw old data away. In fact, it is required to be kept in this application
- records occupy one or more full sectors -- the write once aspect of optical disks requires that every sector used is a full sector, partial sectors cannot be written to and then added to at a later time. So we use full sector records, even if it means adding filler at the end of our data fields.
- post fields -- allow us to update with write once media
- post fields -- allow us an "audit trail" back to previous revisions of data

```

struct keytag {
    char value[20];          /* structure describing the self-sorting key list */
    struct *keytag left;     /* the key word */
    struct *keytag right;    /* the left child of the tree */
    /* the right child of the tree */
};

struct uptag {
    long recnum;             /* structure describing each level of update */
    long datetime;           /* record number */
    /* date and time */
};

struct curtag {
    long recnum;             /* structure describing the current record */
    int updates;             /* record number */
    char *recbody;           /* number of updates made */
    char *prevrec;           /* pointer to the latest data */
    char *search;            /* pointer to a previous version */
    struct uptag[MAXVERS];   /* current search criteria */
    /* list of updates */
};

long Totrecs;               /* total records in database */
long Curecnum;              /* current record being worked on */
struct Currec;              /* current record being worked on */

int Ofd;                    /* the original data records file */
int Ufd;                    /* the updates file */
int Kfd;                    /* the keyword file */

struct emptytag {
    long date;               /* date of this entry */
    char lname[20];          /* last name */
    char fname[20];          /* first name */
    char mname[20];          /* middle name */
    char socsec[8];          /* social security number */
    char address1[20];       /* address line 1 */
    char address2[20];       /* address line 2 */
    char city[20];           /* city */
    char state[2];           /* state */
};

```

(continued)

```

char zip[9];          /* zip code */
char telephone[12];

*/
char birthpl[40];     /* place of birth */
char sex;
char spousnm[40];     /* name of spouse */
char benefic1[40];    /* primary beneficiary */
char benefic2[40];    /* secondary beneficiary */
char title[15];       /* job title / position */
int level;            /* pay level / job grade */
int dept;             /* department */
long salary;          /* current salary */
/* employment progression history */
long hiredate;        /* hiring date */
char filler[to make 512 bytes];
};

/* assumed functions */
/* oopen, oclose, oread, owrite, and olseek work just like their magnetic */
/* stdio counterparts */
/* available from the Optotech Optical Drive Toolkit for PC-DOS */
/* Optotech, 303-570-7500 */

int oopen(fname, mode) /* opens file "fname" in mode on optical disk */
/* returns file descriptor */
/* valid mode are 0, 1, and append (2) */
int oclose(fd) /* closes file fd previously oopened */
int oread(fd, buf, len) /* reads len bytes into buf from fd, returns */
/* number of bytes read */
int owrite(fd, buf, len) /* writes len bytes into buf from fd, returns */
/* number of bytes written */
long olseek(fd, pos, mode) /* seeks to pos in file fd based on mode */
/* mode == 0, from beginning */
/* mode == 1, from current position */
/* mode == 2, from end */
long curdtm(); /* returns current date and time */
int getentry(buffer) /* gets a record and puts it in buffer */
int chgentry(buffer) /* takes the record in buffer, collects changes */
/* to it, then returns the modified buffer */
int owritep(fd, record, value) /* writes the post field of record of file fd */
/* with the location stored in value */
int oreadp(fd, record, &value) /* reads the post field of record of file fd */
/* and fills value with the location*/
updtree(root, record, number) /* adds values to the binary key tree starting */
/* node root, uses number as the reference */
/* location back into the data files */
long keymatch(root, string) /* tries to find a match in the key tree for */
/* string, returning the record number if */
/* successful */
int getkey(keystr) /* gets a search key and puts it into keystr */
int display() /* an error display routine */
int disprec(record) /* displays a record */
int abort(msg) /* displays msg then exits to OS */
int menu() /* displays a list of choices and returns with */
/* selection number */
report() /* runs a report generator */

addentry()
{
char record[RECSIZE];

rtn = getentry(record);
if(!rtn)
return 0;
else {
add(record);
return 1;
}
}

update()
{

```



```

/* get the data */
rtn = chgentry(Currec.recbody);

/* set up the new data fields */
newrec = (struct *emplitag) Currec.recbody;
newrec->date = curdtm();

/* now find out where this record will go in the update file */
recnum = olseek(Ufd, 0L, 2)/RECSIZE;
/* put it there */
owrite(Ufd, newrec, RECSIZE);

/* now, write the previous "latest record's" postfield with */
/* the location (record number) of our recent update */
/* write it in the original file if it's the first update */
if(Currec.updates > 0)
    owritep(Ufd, recnum, Currec.uptag[updates].recnum);
else
    owritep(Ofd, recnum, Currec.recnum);

/* now, update the Currec structure */
Currec.updates++;
Currec.updata[Currec.updates].recnum = recnum;
Currec.updata[Currec.updates].datetime = curdtm();
return 1;
}

getrecr(recnum)          /* gets a relative record */
long recnum;
{
    Currec.updates = 0;

    /* seek to and get the original record */
    olseek(Ofd, recnum*RECSIZE, 0);
    oread(Ofd, Currec.recbody, RECSIZE);

    /* now follow the update chain */
    postfld = oreadp(Ofd, recnum, &postval);
    while(postfld) {
        /* go to the new data and read it in */
        olseek(Ufd, postfld*RECSIZE, 0);
        oread(Ufd, Currec.recbody, RECSIZE);
        /* now update Currec's update list */
        Currec.updata[Currec.updates].recnum = postval;
        temp = (struct *emplitag) Currec.recbody;
        Currec.updata[Currec.updates].datetime = temp.date;
        Currec.updates++;

        /* now, see if there's another one */
        postfld = oreadp(Ufd, postval*RECSIZE, &postval);
    }

    Curecnum = recnum;
    return 1;
}

getold(level)           /* gets a previous revision of the current record */
int level;              /* and puts it in Currec.prevrec */
{
    olseek(Ufd, Currec.updata[level].recnum * RECSIZE, 0);
    oread(Ufd, Currec.prevrec, RECSIZE);
}

getrevs()              /* returns the number of revisions of current record */
{

```

(continued)

```

    return(Currec.updates);
}

gozero()                                /* position at the first record */
{
    getrecr(0L);
}

goend()                                  /* position at the last record */
{
    getrecr(Totrecs - 1L);
}

getnext()                                /* get the next relative record */
{
    if(Curecnum + 1L < Totrecs) {
        getrecr(Curecnum + 1L);
        return 1;
    } else
        return 0;
}

getprev()                                /* get the previous relative record */
{
    if(Curecnum - 1L >= 0) {
        getrecr(Curecnum - 1L);
        return 1;
    } else
        return 0;
}

add(record)                              /* add a record to the database */
char *record;
{
    /* go to the end of the data base */
    recnum = olseek(Ofd, 0L, 2);

    /* now write the data */
    owrite(Ofd, record, RECSIZE);

    /* now update the binary key tree with keywords and record number */
    updtree(Treeroot, record, recnum);

    Totrecs++;
}

long findkey(keystr)                      /* do a key search and return record */
char *keystr;
{
    matchrec = keymatch(Treeroot, keystr);

    if(matchrec == -1L)
        return -1L;
    else {
        /* get the record and return its value */
        getrecr(matchrec);
        return(matchrec);
    }
}

search()                                  /* search the database for a key match */
{

```



```

    getkey(keystr);
    rtn = findkey(keystr);

    /* depending on result, display error or record */
    if(rtn == -1L)
        display("No match found");
    else
        disprec(rtn);
}

startup()                                /* perform database initialization */
{
    /* open the database files to allow for additions */
    Ofd = oopen("1:original.dat", OAPPEND);
    if (Ofd == -1) {
        abort("Couldn't open data file");
    }

    Ufd = oopen("1:update.dat", OAPPEND);
    if (Ufd == -1) {
        oclose(Ofd);
        abort("Couldn't open update file");
    }

    Kfd = oopen("1:keyfile.dat", OAPPEND);
    if (Kfd == -1) {
        oclose(Ofd);
        oclose(Ufd);
        abort("Couldn't open key file");
    }

    /* set total record number */
    Totrecs = olseek(Ofd, 0L, 2)/RECSIZE;

    /* and go to the first record */
    gozero();
}

quit()
{
    oclose(Ofd);
    oclose(Ufd);
    oclose(Kfd);
}

mainmenu()
{
    startup();
    while(1) {
        switch(menu()) {
            case 1:
                addentry();
                break;
            case 2:
                update();
                break;
            case 3:
                search();
                break;
            case 4:
                report();
                break;
            case 5:
                quit();
                break;
        }
    }
}

```

(continued)

```

default:
    error("Not a valid entry, try again");
    break;
}
}
}

```

pell.bas

TEXT
 Mathematical Recreations: "The Pellian Equation," Robert T. Kurosaka.
 May, page 379.

```

10 '*****
20 '*      PELLIAN EQUATION      *
30 '*      X^2 - D*Y^2 = 1      *
40 '*      BY BOB KUROSAKA      *
50 '*****
60 REM ENTER A NONSQUARE INTEGER D. THE PROGRAM
70 REM DETERMINES THE FIRST (LEAST) SOLUTION, THE
80 REM RECURSIVE FORMULAS, AND THE 2ND SOLUTION.
90 REM
100 REM
110 CLS
120 DIM P(100),Q(100),A(100)  'Hope 100 is enough!
130 INPUT "D = ";D
140 D=ABS(INT(D))
150 RD=VAL(STR$(SQR(D)))  'Remove guard digits from SQR(D)
160 IF RD<>INT(RD) THEN 180
170 PRINT "D must not be a square!":GOTO 130
180 '
190 '-----FIRST HALF: FIND As-----
200 '
210 P(1)=0:Q(1)=1:A(1)=INT(SQR(D)):N=1  '1ST COLUMN VALUES
220 N=N+1
230 P(N)=A(N-1)*Q(N-1)-P(N-1)
240 Q(N)=(D-P(N)*P(N))/Q(N-1)
250 A(N)=INT((A(1)+P(N))/Q(N))
260 IF A(N)=2*A(1) THEN 290
270 GOTO 220
280 '
290 '-----SECOND HALF: FIND X0 AND Y0-----
300 '
310 CL=N-1  'CL=CYCLE LENGTH
312 FOR I=N+1 TO 2*CL  'REPEAT CYCLE
314 A(I)=A(I-CL): P(I)=P(I-CL): Q(I)=Q(I-CL)
316 NEXT I
320 IF CL/2<>INT(CL/2) THEN CL=2*CL
330 DIM X(CL), Y(CL)
340 '
350 '-----FIND Xs AND Ys-----
360 '
370 X(1)=A(1):Y(1)=1  'SET UP 1ST 2 COLUMNS
380 X(2)=A(1)*A(2)+1:Y(2)=A(2)  ' OF Xs AND Ys
390 IF CL<3 THEN 450
400 FOR I=3 TO CL
410     X(I)=A(I)*X(I-1)+X(I-2)
420     Y(I)=A(I)*Y(I-1)+Y(I-2)
430 NEXT I
440 '
450 '-----FIRST SOLUTION-----
460 '
470 X0=X(CL):Y0=Y(CL):C=2*X0  '(X0,Y0) = LEAST SOLUTION
480 PRINT:PRINT "LEAST SOLUTION: "
490 PRINT "X = ";X0
500 PRINT "Y = ";Y0
510 '
520 '-----PRINT RECURSIVE FORMULAS-----
530 '
540 PRINT:PRINT "FORMULAS: "
550 PRINT "X(N) = ";C;"* X(N-1) - X(N-2)"
560 PRINT "Y(N) = ";C;"* Y(N-1) - Y(N-2)"

```



```

570 '
580 '-----CHECK X0 AND Y0-----
590 '
600 PRINT:PRINT "CHECK:"
610 L=LEN(STR$(D))
620 PRINT TAB(L+3)"X^2 = ";X0*X0
630 PRINT D;"* Y^2 = ";D*Y0*Y0
640 '
650 '-----PRINT 2ND SOLUTION-----
660 '
670 PRINT:PRINT "SECOND SOLUTION;"
680 PRINT "X = ";C*X0-1;"      Y = ";C*Y0
690 END

```

routine.sub

TEXT

Programming Insight: "Subroutine Overlays in GW-BASIC," Mike Carmichael. May, page 151. Also download mainprog.bas.

```

30000 '
30001 ' *****
30002 ' routine.sub
30003 ' subroutine #1
30004 ' Note: first line must consist of only line # and one 'REM' statement
30005 ' Also, when saving or loading the binary version, 'routine.sub'
30006 ' must be placed at the end of 'mainprog'
30007 '
30008 ' *****
30009 '
30010 CLS
30020 GOSUB 12000: ' heading again for the fun of it
30030 LOCATE 16,20: PRINT "subroutine #1";
30040 RETURN
31000 ' *****
31001 ' subroutine #2
31002 '
31003 '
31004 '
31010 LOCATE 17,20: PRINT "subroutine #2";
31020 LOCATE 18,20: PRINT "returning to main program";
31030 RETURN

```


June

indexbpp.lst

TEXT

Programming Project: "A Simple File-Indexing System," by Bruce Webster.
June, page 92. Also download indexbpp.pas.

```
const
  IndexMax      = 1000;
  RecCountErr   = -2;
  NewFileCreated = -1;
  NoError       = 0;
  RecordNotFound = 1;
  NoMoreRoom    = 2;
  AlreadyExists = 3;
  OutOfRange    = 4;

type
  Keytype      = string[40];
  FileStr      = string[80];

  DataRec = record
    case Boolean of
      True   : (NumRecs : Integer);
      False  : (Key      : Keytype;
                theRest  : Whatever;
                { this represents the rest of your data fields } );
    end;

  IndexRec = record
    Key   : Keytype;
    Num   : Integer;
  end;

  IndexList = array[1..IndexMax] of IndexRec;

var
  KList      : IndexList;
  DFile      : file of DataRec;
  MaxRec     : Integer;
```

LISTING 1. Global definitions and declarations.

```
{
  compiler-specific file I/O routines
  these procedures are specific to TURBO Pascal. If you
  are using another Pascal compiler, you will need to
  modify them appropriately. Note that TURBO Pascal does
  not support the standard routines GET and PUT, but instead
  uses READ and WRITE.
}

{$I-} { turn off I/O error checking }

procedure FRead(RNum : Integer; var Rec : DataRec; var Error : Integer);
{
  reads record #RNum into Rec
}
begin
  if (RNum < 0) or (RNum > MaxRec)
  then Error := OutOfRange
  else begin
    Seek(DFile, RNum);
    Error := IOResult;
    if Error = NoError then begin
      Read(DFile, Rec);
      Error := IOResult
    end;
  end;
end;
```

(continued)

```

end;
if Error > 0
then Error := 100 + Error
end
end; { of proc FRead }

procedure FWrite(RNum : Integer; Rec : DataRec; var Error : Integer);
{
    writes record #RNum into Rec
}
begin
    if (RNum < 0) or (RNum > MaxRec)
    then Error := OutOfRange
    else begin
        Seek(DFile,RNum);
        Error := IOResult;
        if Error = NoError then begin
            Write(DFile,Rec);
            Error := IOResult
        end;
        if Error > 0
        then Error := 100 + Error
    end
end; { of proc FRead }

procedure FOpen(FileName : FileStr; var Error : Integer);
{
    tries to open FileName; if it doesn't exist, creates
    it with the appropriate header record
}
const
    TurboNoFile = 1; { "no file" error code for TURBO Pascal }
var
    IOCode      : Integer;
    TRec        : DataRec;
begin
    Assign(DFile,FileName);
    Reset(DFile);
    IOCode := IOResult;
    if IOCode = TurboNoFile then begin { file doesn't exist }
        FillChar(TRec,SizeOf(TRec),0);
        Rewrite(DFile);
        TRec.NumRecs := 0;
        FWrite(0,TRec,Error);
        Close(DFile);
        Assign(DFile,Filename);
        Reset(DFile);
        IOCode := IOResult;
        if IOCode = NoError
        then Error := NewFileCreated
    end;
    if IOCode <> NoError
    then Error := 100 + IOCode;
end; { of proc FOpen }

procedure FClose(var Error : Integer);
{
    closes file
}
begin
    Close(DFile);
    Error := IOResult;
    if Error > 0
    then Error := Error + 100
end; { of proc FClose }

{$I+} { turn on I/O error checking }

```

LISTING 2a. File I/O routines specific to TURBO Pascal.

```

{
    compiler-specific file I/O routines
    these procedures are specific to UCSD Pascal. If you
    are using another Pascal compiler, you will need to
    modify them appropriately.
}

```



```

{$I-} { turn off I/O error checking }

procedure FRead(RNum : Integer; var Rec : DataRec; var Error : Integer);
{
    reads record #RNum into Rec
}
begin
    if (RNum < 0) or (RNum > MaxRec)
    then Error := OutOfRange
    else begin
        Seek(DFile,RNum);
        Error := IOResult;
        if Error = NoError then begin
            Get(DFile);
            Error := IOResult;
            if Error = NoError
            then Rec := DFile^
            end;
            if Error <> NoError
            then Error := 100 + Error
        end
    end;
end; { of proc FRead }

procedure FWrite(RNum : Integer; Rec : DataRec; var Error : Integer);
{
    writes record #RNum into Rec
}
begin
    if (RNum < 0) or (RNum > MaxRec)
    then Error := OutOfRange
    else begin
        Seek(DFile,RNum);
        Error := IOResult;
        if Error = NoError then begin
            DFile^ := Rec;
            Put(DFile);
            Error := IOResult
        end;
        if Error > 0
        then Error := 100 + Error
    end
end; { of proc FRead }

procedure FOpen(FileName : FileStr; var Error : Integer);
{
    tries to open FileName; if it doesn't exist, creates
    it with the appropriate header record
}
const
    UCSDNoFile = 1; { "no file" error code for UCSD Pascal }
var
    IOCode      : Integer;
    TRec        : DataRec;
begin
    Reset(DFile,FileName);
    IOCode := IOResult;
    if IOCode = UCSDNoFile then begin { file doesn't exist }
        FillChar(TRec,SizeOf(TRec),Chr(0));
        Rewrite(DFile,FileName);
        TRec.NumRecs := 0;
        FWrite(0,TRec,Error);
        Close(DFile,Lock);
        Reset(DFile,FileName);
        IOCode := IOResult;
        if IOCode = NoError
        then Error := NewFileCreated
    end;
    if IOCode <> NoError
    then Error := 100 + IOCode;
end; { of proc FOpen }

procedure FClose(var Error : Integer);
{
    closes file
}

```

(continued)

June

```
begin
  Close(DFile,Lock);
  Error := IOResult;
  if Error > 0
    then Error := Error + 100
end; { of proc FClose }

{$I+} { turn on I/O error checking }
```

LISTING 2b. File I/O routines specific to UCSD Pascal.

```
-----

procedure SortIndexList;
{
  sorts the array KList using a selection sort technique
}
var
  I,J,Min      : Integer;
  Temp         : IndexRec;
begin
  for I := 1 to MaxRec-1 do begin
    Min := I;
    for J := I+1 to MaxRec do
      if KList[J].Key < KList[Min].Key
        then Min := J;
    Temp := KList[I];
    KList[I] := KList[Min];
    KList[Min] := Temp
  end
end; { of proc SortIndexList }

procedure InitStuff(FileName : FileStr; var Error : Integer);
{
  sets everything up for indexing system. This assumes that
  there are no more than IndexMax (=1000) records, and that the
  records are numbered 1..IndexMax. Record #0 is the header
  record and is used to store the current number of records
  actively being used in the file
}
var
  Indx,TErr      : Integer;
  TRec           : DataRec;
begin
  Error := NoError;
  FOpen(FileName,Error);
  if Error <= NoError then begin
    MaxRec := 0;
    FRead(0,TRec,TErr);
    Error := TErr;
    MaxRec := TRec.NumRecs;
    for Indx := 1 to MaxRec do begin
      FRead(Indx,TRec,TErr);
      if TErr > 0
        then Error := TErr;
      KList[Indx].Key := TRec.Key;
      KList[Indx].Num := Indx
    end;
    SortIndexList
  end
end; { of proc InitStuff }

procedure CleanupStuff(var Error : Integer);
{
  this just does an orderly shutdown and should be called
  before you leave your program (or open another data file)
}
var
  TRec           : DataRec;
begin
  TRec.NumRecs := MaxRec; { save out # of records }
  FWrite(0,TRec,Error);
  FClose(Error)
end; { of proc CleanupStuff }
```

LISTING 3. Initialization and cleanup routines.


```

function FindKey(Key : Keytype) : Integer;
{
    looks for Key in KList; returns location in KList
    if found; otherwise returns - 1
}
var
    L,R,Mid      : Integer;
begin
    L := 1; R := MaxRec;
    repeat
        Mid := (L+R) div 2;
        if Key < KList[Mid].Key
            then R := Mid-1
            else L := Mid+1
    until (Key = KList[Mid].Key) or (L > R);
    if Key = KList[Mid].Key
        then FindKey := Mid
        else FindKey := -1
end; { of proc FindKey }

procedure GetRecord(Key : Keytype; var Rec : DataRec;
    var Error : Integer);
{
    looks through KList for Key; if found, returns in Rec.
    It and the routines that follow assume the procedure Seek
    for random access of the file of records.
}
var
    Item          : Integer;
begin
    Error := NoError;
    Item := FindKey(Key);
    if Item > 0
        then FRead(KList[Item].Num,Rec,Error)
        else Error := RecordNotFound
end; { of proc GetRecord }

procedure PutRecord(Rec : DataRec; var Error : Integer);
{
    writes Rec out to the file. If a record with that
    key already exists, then overwrites that record;
    otherwise, adds the record to the end of the file.
    If there's no more room for records, exits with an
    error code
}
var
    Item          : Integer;
begin
    Error := NoError;
    Item := FindKey(Rec.Key);
    if Item >= 0
        then FWrite(KList[Item].Num,Rec,Error)
    else if MaxRec < IndexMax then begin
        MaxRec := MaxRec + 1;
        FWrite(MaxRec,Rec,Error);
        KList[MaxRec].Key := Rec.Key;
        KList[MaxRec].Num := MaxRec;
        SortIndexList
    end
    else Error := NoMoreRoom
end; { of proc PutRecord }

```

LISTING 4. Basic record access routines.

```

procedure AddRecord(Rec : DataRec; var Error : Integer);
{
    adds a record to the file. If a record with the same
    key already exists, then exits with an error code
}
var
    Item          : Integer;
begin
    Error := NoError;

```

(continued)

```

Item := FindKey(Rec.Key);
if Item > 0
then Error := AlreadyExists
else PutRecord(Rec,Error)
end; { of proc AddRecord }

procedure DeleteRecord(Key : Keytype; var Error : Integer);
{
  deletes the record with 'Key' by copying the last record
  in the file to that slot, then modifies KList by shuffling
  all the key entries up
}
var
  Item,Last,Max,MVal      : Integer;
  TRec                   : DataRec;
begin
  Error := NoError;
  Item := FindKey(Key);
  if Item = -1
  then Error := RecordNotFound
  else begin
    Max := 1; MVal := KList[Max].Num;
    for Last := 2 to MaxRec do
      if KList[Last].Num > MVal then begin
        Max := Last; MVal := KList[Last].Num
      end;
    if Max <> Item then begin
      FRead(MVal,TRec,Error); { get last record in file }
      FWrite(KList[Item].Num,TRec,Error); { write over it }
      KList[Max].Num := KList[Item].Num
    end;
    for Last := Item to MaxRec-1 do { delete KList[Item] }
      KList[Last] := KList[Last+1];
    MaxRec := MaxRec - 1 { adjust # of records }
  end
end; { of proc DeleteRecord }

```

LISTING 5. Higher-level record access routines.

hilbert.bas

TEXT

Programming Insight: "Hilbert Curves Made Simple," by Michael Ackerman.
June, page 137.

```

1 GOTO 1000
2 REM *****
3 REM *          HILBERT          *
4 REM *          *                *
5 REM *  BY MICHAEL ACKERMAN  *
6 REM *          *                *
7 REM *          8/27/85          *
8 REM *****
100 RDER = RDER - 1
110 TURN = - TURN
120 TEMP = DY:DY = - TURN * DX:DX = TURN * TEMP
130 IF RDER > 0 THEN GOSUB 100
140 X = X + DX:Y = Y + DY: HPLLOT TO X,Y
150 TURN = - TURN
160 TEMP = DY:DY = - TURN * DX:DX = TURN * TEMP
170 IF RDER > 0 THEN GOSUB 100
180 X = X + DX:Y = Y + DY: HPLLOT TO X,Y
190 IF RDER > 0 THEN GOSUB 100
200 TEMP = DY:DY = - TURN * DX:DX = TURN * TEMP
210 TURN = - TURN
220 X = X + DX:Y = Y + DY: HPLLOT TO X,Y
230 IF RDER > 0 THEN GOSUB 100
240 TEMP = DY:DY = - TURN * DX:DX = TURN * TEMP
250 TURN = - TURN
260 RDER = RDER + 1
270 RETURN
1000 TEXT : HGR : HCOLOR= 3: INPUT"ORDER <1-7>";RDER

```



```

1010 POKE 49234,1
1020 DY = 192 / 2 ^ RDER
1030 TURN = - 1
1040 DX = X = Y = 0
1050 HPLOT X,Y
1060 GOSUB 100
1070 END

```

fractal.lib

TEXT

"Musical Fractals," Charles Dodge and Curtis R. Bahn.
June, page 185. All the programs mentioned in one TEXT file.

```

1 REM WHITE.BAS is in MSX BASIC with MUSIC MACRO
2 REM commands for the Yamaha CX5-M music computer
10 _INIT:_INST(1)
20 X=RND(-TIME)
30 FOR X = 1 TO 25
35 REM notes in range of 25 to 120
40 N = INT(RND(1)*95)+25
45 REM lengths in range of 1 to 4
50 L = INT(RND(1)*4)+1
60 _PHRASE(1,"L=L;","N=N;")
70 NEXT X
80 _PLAY(1,1)
90 _WAIT(1)
100 INPUT"AGAIN";DD:GOTO 80

```

```

1 REM BROWN.BAS is in MSX BASIC with MUSIC MACRO
2 REM commands for the Yamaha CX5-M music computer
10 _INIT:_INST(1)
20 X=RND(-TIME)
30 N=60:L=2
40 FOR X = 1 TO 25
45 REM R varies the range of the distribution
50 R=3:GOSUB 130:N=N+D
60 IF N>120 OR N<25 THEN N=N-2*D
70 R=.667:GOSUB 130:L=L+D
80 IF L<1 OR L>4 THEN L=L-2*D
90 _PHRASE(1,"L=L;","N=N;")
100 NEXT X
110 _PLAY(1,1):_WAIT(1)
120 INPUT"AGAIN";DD:GOTO 110
130 REM BROWNIAN ROUTINE
140 S=0:REM S is Sum
150 FOR I = 1 TO 12
160 S=S+RND(1)
170 NEXT I
180 D=INT(R*(S-6))
190 RETURN

```

```

1 REM 10VERF.BAS is in MSX BASIC with MUSIC MACRO
2 REM commands for the Yamaha CX5-M music computer
10 _INIT:_INST(1):LL=8:LN=16:S=60:X=RND(-TIME)
20 FOR X = 1 TO 25
30 D=N:GOSUB 130
40 N=D:SN=N+S
50 D=L:GOSUB 130
60 L=D:SL=LL+1
70 _PHRASE(1,"L=SL;","N=SN;")
80 NEXT X
90 _PLAY(1,1)
100 _WAIT(1)
110 INPUT"AGAIN";DD
120 GOTO 90
130 REM 1/F ROUTINE
135 REM L is last value. K is 1/2 poss values. PROBIT=1/K
140 L=D:D=0:K=16:PROBIT=.03125
150 J=INT(L/K)

```

(continued)

```

160 IF J=1 THEN L=L-K
170 U=RND(1)
180 IF U < PROBIT THEN J=1-J
190 D=D+J*K
200 K=K/2
210 PROBIT=PROBIT*2
220 IF K>1 THEN GOTO 150
230 RETURN

```

```

1 REM VARIATN.BAS is in MSX BASIC with MUSIC MACRO
2 REM commands for the Yamaha CX5-M music computer
4 REM
5 REM*****
6 REM Copyright 1986 Curtis Bahn, Creative Associates Inc.
7 REM*****
8 REM
10 CLS
20 DIM P(6):DIM AP(6):DIM BP(36):DIM CP(216)
30 DIM D(6):DIM AD(6):DIM BD(36):DIM CD(216)
40 DT=0:CC=0:BC=0:AC=0:GP=0:R=0:HP=100
50 _INIT:_INST(1):_INST(2):_INST(3)
100 INPUT"HOW MANY NOTES IN SET?"; PN:IF PN>6 OR PN<1 THEN GOTO 100
110 PRINT"INPUT";PN;"PITCH RELATIONSHIPS"
120 FOR LOOP=1 TO PN
130 INPUT P(LOOP):IF ABS(P(LOOP))>12 THEN PRINT"TOO BIG":GOTO 130
135 IF GP<P(LOOP) THEN GP=P(LOOP):IF BP>P(LOOP) THEN BP=P(LOOP)
140 NEXT LOOP
150 PRINT "INPUT";PN;"TIME RELATIONSHIPS"
160 FOR LOOP=1 TO PN
170 INPUT D(LOOP)
180 NEXT
185 INPUT"BROWNIAN RANDOMIZER APPLIED TO PITCH (1 OR 0)";R:IF R>2 GOTO 185
190 PP=ABS(BP)+ABS(GP)
195 LP=HP-(3*GP):SK=100/(3*PP)
200 REM FRACTAL ROUTINE
205 PRINT"COMPUTING FRACTAL"
210 FOR A=1 TO PN
220 AP(A)=P(A)+RC:AD(A)=D(A)
230 FOR B=1 TO PN
240 BC=BC+1:IF R = 1 THEN GOSUB 700
245 BP(BC)=AP(A)+P(B)+RC:BD(BC)=D(B)*D(A)
250 FOR C=1 TO PN
260 CC=CC+1:IF R=1 THEN GOSUB 700
270 CP(CC)=BP(BC)+P(C)+RC:CD(CC)=D(C)*BD(BC):DT=DT+CD(CC)
280 NEXT C: NEXT B: NEXT A
290 TS=255/DT
300 REM PLAYING ROUTINE
310 BC=0:CC=0
320 FOR A=1 TO PN
330 _SOUND(1,1,AP(A)+LP):CIRCLE(TC,90-(AP(A)*SK)),6
340 FOR B= 1 TO PN
345 BC=BC+1
350 _SOUND(2,1,BP(BC)+LP):CIRCLE(TC,90-(BP(BC)*SK)),3
360 FOR C= 1 TO PN
370 _SOUND(3,1,CP(CC)+LP):CIRCLE(TC,90-(CP(CC)*SK)),1
380 FOR LOOP=1 TO CD(CC):TC=TC+TS
385 REM all play statements here for mono playback
390 _SOUND(3,0,CP(CC)+LP)
400 NEXT LOOP
410 NEXT C: NEXT B: NEXT A
420 _STOP(1): _STOP(2): _STOP(3)
430 INKEY$=DD$:IF DD$="" THEN GOTO 430
440 GOTO 300
500 REM BROWNIAN ROUTINE
510 S=0
520 FOR I= 1 TO 12
530 S=S+RND(1)
540 NEXT I
550 RC=INT(2*(S-6))
560 RETURN

```

```

1 REM RANDOM.BAS is in MSX BASIC with MUSIC MACRO
2 REM commands for the Yamaha CX5-M music computer
4 REM

```



```

5 REM*****
6 REM Copyright 1986 Charles Dodge, North Cape Music
7 REM*****
8 REM
9 CLS
10 DIM AP(10):DIM BP(100):DIM CP(255):DIM DP(2,255):TC=20
11 DIM BD(10):DIM CD(100):DIM DD(255)
12 DT=0:DC=0:CC=0:BC=0:AC=0:DIM CT(4)
13 DIM CF(4,12):DIM LAST(4):KN=50
14 X=RND(-TIME):LAST(1)=INT(RND(1)*32)
15 _INIT:_INST(1):_INST(2):_INST(3):_INST(4)
16 _MODI(1,5):_MODI(2,40):_MODI(3,15):_MODI(4,16)
17 FOR LOOP = 1 TO 4
18 PRINT"ENTER PITCH CLASS LIMIT OF LEVEL #";LOOP
19 INPUT PL(LOOP)
20 IF PL(LOOP)>6 THEN PRINT"TOO BIG":GOTO 120
21 IF PL(LOOP)<1 THEN PRINT"TOO SMALL":GOTO 120
22 NEXT LOOP
23 L=1
24 FOR A = 1 TO 10
25 GOSUB 820
26 IF CT(L)>PL(L) THEN GOTO 230
27 AC=AC+1
28 AP(A)=LAST(L)
29 NEXT
30 REM FRACTAL ROUTINE
31 SCREEN 2
32 FOR A = 1 TO AC
33 CIRCLE(DI/2,197-(AP(A)*5+20)),9
34 LAST(2)=AP(A)
35 FOR B = 1 TO 10
36 L=2
37 GOSUB 820
38 IF CT(L)>PL(L) THEN GOTO 570
39 BI=BI+1:BP(BI)=LAST(L)
40 CIRCLE(DI/2,197-(BP(BI)*5+20)),6
41 LAST(3)=BP(BI)
42 FOR C = 1 TO 10
43 L=3
44 GOSUB 820
45 IF CT(L)>PL(L) THEN GOTO 550
46 CI=CI+1:CP(CI)=LAST(L):IF CI=255 THEN AC=A:GOTO 590
47 CIRCLE(DI/2,197-(CP(CI)*5+20)),3
48 LAST(4)=CP(CI)
49 FOR D = 1 TO 10
50 L=4
51 GOSUB 820
52 IF CT(L)>PL(L) THEN GOTO 530
53 DI=DI+1
54 CIRCLE(DI/2,197-(LAST(L)*5+20)),.5
55 IF DI>255 THEN GOTO 500
56 DP(1,DI)=LAST(L):GOTO 520
57 DP(2,DI-255)=LAST(L)
58 IF DI=510 THEN AC = A: GOTO 590
59 DC=DC+1:NEXT D
60 DD(CI)=DC:DC=0:CT(L)=0:GOSUB 1020
61 CC=CC+1:NEXT C
62 CD(BI)=CC:CC=0:CT(L)=0:GOSUB 1020
63 BC=BC+1:NEXT B
64 BD(A)=BC:BC=0:CT(L)=0:GOSUB 1020
65 NEXT A
66 LINE (0,0)-(255,0):LINE (255,0)-(255,197):
67 LINE(255,197)-(0,197):LINE(0,197)-(0,0)
68 DD$=INKEY$:IF DD$="" GOTO 600
69 REM PLAY LOOPS
70 FOR A = 1 TO AC
71 _SOUND(1,1,AP(A)+KN)
72 FOR B = 1 TO BD(A):BC=BC+1
73 _SOUND(2,1,BD(BC)+KN)
74 FOR C = 1 TO CD(B):CC=CC+1
75 IF CC>255 GOTO 770
76 _SOUND(3,1,(CP(CC)+KN)
77 FOR D = 1 TO DD(C):DC=DC+1
78 IF DC>255 GOTO 740
79 _SOUND(4,1,DP(1,DC)+KN):GOTO 760
80 _SOUND(4,1,DP(2,DC-255)+KN)

```

(continued)

```

750 IF DC=510 THEN GOTO 770
760 NEXT D:NEXT C:NEXT B:NEXT A
770 _STOP(1):_STOP(2):_STOP(3):_STOP(4)
780 DD$=INKEY$:IF DD$="" GOTO 780
790 BC=0:CC=0:DC=0
800 GOTO 610
820 REM 1/F ROUTINE
830 LL=LAST(L):NP=0:K=16:PROBIT=.03125
840 J=INT(LL/K)
850 IF J=1 THEN LL=LL-K
860 U=RND(1)
870 IF U<PROBIT THEN J=1-J
880 NP=NP+J*K
890 K=K/2
900 PROBIT = PROBIT*2
910 IF K>=1 GOTO 840
920 LAST(L)=NP:TEST=NP
930 REM PITCH CLASS TEST
940 FOR I = 0 TO 11
950 IF INT((TEST+I)/12)=(TEST+I)/12
    THEN CF(L,I)=1:GOTO 920
960 NEXT I
970 CT(L)=0
980 FOR I = 0 TO 11
990 CT(L)=CF(L,I)+CT(L)
1000 NEXT I
1010 RETURN

```

midi.arc

BINARY

"MIDI Programming" by Donald Swearingen.

June, page 211. All the programs from the article. Requires ARC.

=====
LIST1.PAS
=====

```

const
  TIMING_OVERFLOW = 248;      { MPU-401 Constants }
  NOP = 248;
  MEASURE_END = 249;
  DATA_END = 252;
  MAX_TIMING_COUNT = 240;

  TIMEBASE = 120.0;          { MPU-401 Default Timebase }
  TEMPO = 100.0;             { MPU-401 Default Tempo }

  MIN_MIDI_DATA = 0;         { Minimum MIDI Data Value }
  MAX_MIDI_DATA = 127;       { Maximum MIDI Data Value }

  NOTE_OFF = 0;              { MIDI Commands }
  NOTE_ON = 1;
  AFTER_TOUCH_K = 2;
  CONTROL_CHANGE = 3;
  PROGRAM_CHANGE = 4;
  AFTER_TOUCH_P = 5;
  PITCH_WHEEL = 6;
  SYSTEM_EXCLUSIVE = 7;

  MIDI_MESS_TEXT :           { MIDI Command Text Strings }
    array[0..7] of string[20] =
    ('Note Off',
     'Note On',
     'After Touch (key)',
     'Control Change',
     'Program Change',
     'After Touch (poly)',
     'Pitch Wheel',
     'System Exclusive');

  ERR = -1;                  { Function error flags }
  NOERR = 0;

```



```

TRACK_DATAFILE_SIZE = 4096; { MPU-401 track data file }

FILENAME_LEN = 14;           { MSDOS filename length }
RECORD_LEN = 128;           { MSDOS record length }

DIGITS :                     { Hex conversion digits }
  array[0..15] of char =
    '0123456789ABCDEF';

```

```
=====
```

```
LIST2.pas
```

```
=====
```

```

type
  hex_str = string[2];      { Result of byte-->hex conversion }
  track_event_type =        { MPU-401 Track event type }
  (
    OVFL,                    { Timing Overflow }
    MARK,                    { MPU mark }
    MIDI,                    { MIDI using curr. running status }
    MIDI_RS,                 { MIDI setting new running status }
    UNKNOWN                  { Undefined track event }
  );
  track_event =              { Single track event }
  record
    time : byte;             { Event relative time }
    mess : array[1..3] of byte; { Event directive }
  end;
  track_event_block =        { Track event access environment }
  record
    running_status : byte; { Current track running status }
    event_len : 1..4; { Event length, including timing byte }
    event_type : track_event_type;
    event : track_event;
  end;
  track_data_stream =        { In memory track data stream file }
  array[1..TRACK_DATAFILE_SIZE] of byte;
  track_data_block = { track data stream access environment }
  record
    tds : track_data_stream; { track data }
    tds_ptr : 1..TRACK_DATAFILE_SIZE; { track data read pointer }
    edat : boolean; { indicates end of track data }
    curr : track_event_block; { current track event }
  end;

```

```
const
```

```

                                { Track overflow event constant }
OVFL_EVENT : track_event_block =
  (
    running_status:0;          { used to insert timing spacers }
    event_len:1;               { into Track Data Stream }
    event_type:OVFL;
    event:
      (
        time:MAX_TIMING_COUNT;
        mess:(0,0,0)
      )
  );

```

```
=====
```

```
LIST3.PAS
```

```
=====
```

```

{ Return true if input is a MIDI status byte }
function midi_status(midi_data_byte:byte):boolean;
begin
  if (midi_data_byte > MAX_MIDI_DATA) then
    midi_status:=true
  else

```

(continued)

June

```
    midi_status:=false;
end;

{ Return the channel # from a MIDI status byte
}
function midi_chan(running_status:byte):byte;
begin
    midi_chan:=running_status and 15;
end;

{ Return the command portion of a MIDI status byte
}
function midi_cmnd(running_status:byte):byte;
begin
    midi_cmnd:=(running_status shr 4) and 7;
end;

{ Return # of data bytes associated
  with a given MIDI status byte
}
function nm-dat(running_status:byte):byte;
begin
    if (midi_cmnd(running_status) in
        [PROGRAM_CHANGE, AFTER_TOUCH_P]) then
        nm-dat:=1
    else
        nm-dat:=2;
    end;
end;

{ Limit input to valid MIDI data range
}
function midi_data_limit(midi_data_byte:integer):byte;
begin
    if midi_data_byte < MIN_MIDI_DATA then
        midi_data_limit:=MIN_MIDI_DATA
    else if midi_data_byte > MAX_MIDI_DATA then
        midi_data_limit:=MAX_MIDI_DATA
    else
        midi_data_limit:=midi_data_byte;
    end;
end;

====
LIST4.PAS
====

{ Reset status and pointer variables in track data block
}
procedure reset_track_data(var tdt:track_data_block);
begin
    with tdt do
        begin
            tds_ptr:=1;
            edat:=false;
            curr.running_status:=0;
            curr.event_type:=UNKNOWN;
        end;
    end;

{ Load track data stream from user
  specified file into track data block
}
procedure load_track_data(var tdt:track_data_block);
var
    tdf : File;
    tdfn : string[FILENAME_LEN];
begin
    reset_track_data(tdt);
    write('Track data filename: ');
    readln(tdfn);
    assign(tdf,tdfn);
    reset(tdf);
    blockread(tdf,tdt.tds,TRACK_DATAFILE_SIZE div RECORD_LEN);
    close(tdf);
```



```

end;

{ Save track data stream from track
  data block to user specified file
}
procedure save_track_data(tdt:track_data_block);
var
  tdf : File;
  tdfn : string[FILENAME_LEN];
begin
  write('Track data filename: ');
  readln(tdfn);
  assign(tdf,tdfn);
  rewrite(tdf);
  blockwrite(tdf,tdt.tds,TRACK_DATAFILE_SIZE div RECORD_LEN);
  close(tdf);
end;

{ Return current track data byte from track data block
}
function this_byte(tdt:track_data_block):byte;
begin
  this_byte:=tdt.tds[tdt.tds_ptr];
end;

{ Advance pointer to next track data byte
  in track data block
}
procedure advance(var tdt:track_data_block);
begin
  tdt.tds_ptr:=tdt.tds_ptr+1;
end;

{ Convert byte to hexadecimal ASCII string
}
function itox(i:byte): hex_str;
begin
  itox[0]:=chr(2);
  itox[1]:=DIGITS[i div 16];
  itox[2]:=DIGITS[i mod 16];
end;

{ Dump track data stream in hexadecimal format
}
procedure dump_track_data(var tdt:track_data_block);
label
  return;
var
  n,st,off : Integer;
begin
  writeln('Track Data Stream Dump...');
  writeln;
  write(' ');
  for n:=0 to 15 do
    write(itox(n):4);
  writeln;
  n:=0;
  while (n < TRACK_DATAFILE_SIZE div 16) do
    begin
      st:=n*16;
      write(itox(st div 256):2,itox(st mod 256):2,' ');
      for off:=0 to 15 do
        begin
          write(itox(ord(tdt.tds[st+off+1])):4);
          if (tdt.tds[st+off+1] = DATA_END) then
            goto return;
          end;
        write(n);
        n:=n+1;
      end;
    return;
  writeln;
end;

```

(continued)

June

====
LIST5.PAS
====

```
{ Fill in the message bytes of the current
  Track Event in a Track Data Block
}
procedure track_event_message(var tdt:track_data_block);
var
  i : byte; { index counter }
label
  return;
begin
  with tdt.curr do
    begin
      case this_byte(tdt) of
        NOP, MEASURE_END, DATA_END:
          begin
            event_type:=MARK;
            if (this_byte(tdt) = DATA_END) then
              tdt.edat:=true;
            event.mess[event_len]:=this_byte(tdt);
            event_len:=event_len+1;
            advance(tdt);
            goto return;
          end;
        128..239: { MIDI status byte }
          begin
            running_status:=this_byte(tdt);
            event_type:=MIDI_RS;
            event.mess[event_len]:=this_byte(tdt);
            event_len:=event_len+1;
            advance(tdt);
          end;
        else
          event_type:=MIDI;
        end; { case }

      { fill in MIDI data bytes }
      for i:=1 to nmdat(tdt.curr.running_status) do
        begin
          event.mess[event_len]:=this_byte(tdt);
          event_len:=event_len+1;
          advance(tdt);
        end;
      end; { with tdt.curr }
    end;
  return;
end;

{ Advance to the next Track Event in a Track Data Block
}
procedure next_track_event(var tdt:track_data_block);
label
  return;
begin
  if (tdt.edat) then { end of data }
    goto return;
  with tdt.curr do
    begin
      event_len:=1; { count event time }
      case this_byte(tdt) of
        TIMING_OVERFLOW:
          begin
            event_type:=OVFL;
            event.time:=MAX_TIMING_COUNT;
            advance(tdt);
            goto return;
          end;
        0..239: { timing byte }
          begin
            event.time:=this_byte(tdt);
            advance(tdt);
            track_event_message(tdt);
          end;
        end; { case }
      end; { with tdt.curr }
    end;
  return;
end;
```



```
return;
end;
```

```
{ Store a Track Event in a designated Track Data Block
```

```
procedure store_track_event(var tdo:track_data_block;
                           eblk:track_event_block);
```

```
var
  i : byte; { index counter }
begin
  case eblk.event.time of
    MAX_TIMING_COUNT:
      begin
        tdo.tds[tdo.tds_ptr]:=TIMING_OVERFLOW;
        advance(tdo);
      end;
    0..239:
      begin
        tdo.tds[tdo.tds_ptr]:=ebk.event.time;
        advance(tdo);
        for i:=1 to eblk.event_len - 1 do
          begin
            tdo.tds[tdo.tds_ptr]:=ebk.event.mess[i];
            advance(tdo);
          end;
        end;
      end;
  end; { case }
end;
```

```
{ Display a track event on the user console
```

```
procedure disp_event(eblk:track_event_block);
```

```
var
  i : byte; { index counter }
label return;
begin
  with eblk do
    begin
      write(event.time:4);
      if (event_len = 1) then
        begin
          write(' Timing Overflow':16);
          goto return;
        end;
      if (event.mess[1] in [NOP,MEASURE_END,DATA_END]) then
        begin
          case event.mess[1] of
            NOP :
              begin
                write('NOP':16);
                goto return;
              end;
            MEASURE_END:
              begin
                write('Measure End':16);
                goto return;
              end;
            DATA_END:
              begin
                write('Data End':16);
                goto return;
              end;
          end; {case}
        end; {if}
      i:=1;
      if (midi_status(event.mess[1])) then
        begin
          write(MIDI_MESS_TEXT[midi_cmnd(event.mess[1]):16);
          i:=i+1;
        end
      else
        write(' ':16);
      while (i <= (event_len - 1)) do
        begin
          write(event.mess[i]:4);
          i:=i+1;
        end;
    end;
  end;
```

(continued)

June

```
    end;
    end; { with eblk }
return;
writeln;
end;

{ Display all of the Track Events in a Track Data Block
}
procedure disp_track_data(var tdt:track_data_block);
var
    time : real; { Actual time of current track event }
begin
    time:=0.0;
    reset_track_data(tdt);
    while not(tdt.edat) do
        begin
            next_track_event(tdt);
            time:=time+tdt.curr.event.time;
            write( ((time*60)/(TIMEBASE*TEMPO)):8:3 );
            disp_event(tdt.curr);
        end;
    end;
end;
```

```
====
LIST6.PAS
====
```

```
{ Return offset of MIDI key data in
  Track Event message, if present
}
function midi_key_offset(var eblk:track_event_block;
                        chan:byte):integer;
begin
    midi_key_offset:=ERR; { default return value }
    with eblk do
        begin
            if ((event_type in [MIDI, MIDI_RS])
                and (midi_cmnd(running_status)
                    in [NOTE_OFF, NOTE_ON, AFTER_TOUCH_K])
                and (midi_chan(running_status) = chan)) then
                begin
                    midi_key_offset:=1;
                    if (event_type = MIDI_RS) then
                        midi_key_offset:=2;
                    end;
                end;
        end; { with eblk }
    end;

{ Return MIDI key data from Track Event, if present
}
function get_midi_key(var eblk:track_event_block;
                    chan:byte):integer;
var
    key_offset : integer;
begin
    get_midi_key:=ERR; { default return value }
    key_offset:=midi_key_offset(eblk,chan);
    if (key_offset <> ERR) then
        get_midi_key:=eblock.event.mess[key_offset];
    end;

{ Set MIDI key value in a Track Event,
  if Track Event is of appropriate type
}
procedure set_midi_key(var eblk:track_event_block;
                    chan,key:byte);
var
    key_offset : integer;
begin
    key_offset:=midi_key_offset(eblk,chan);
    if (key_offset <> ERR) then
        eblock.event.mess[key_offset]:=key;
    end;
```



```

{ Transpose all MIDI pitch (key) data
  for a channel in a Track Data block
}
procedure transpose_pitch(var tdi,tdo:track_data_block;
                          chan,trans:integer);
var
  curr_key : integer; { MIDI key value from
                       current track event }
begin
  reset_track_data(tdi);
  reset_track_data(tdo);
  while not(tdi.edat) do
    begin
      next_track_event(tdi);
      curr_key:=get_midi_key(tdi.curr,chan);
      if (curr_key <> ERR) then
        set_midi_key(tdi.curr,chan,
                     midi_data_limit(curr_key+trans));
      store_track_event(tdo,tdi.curr);
    end;
  end;
end;

```

```

====
LIST7.PAS
====

```

```

{ Return offset of MIDI velocity data in
  Track Event message, if present
}
function midi_vel_offset(eblk:track_event_block;
                        chan:byte):integer;
var
  offset : integer;
begin
  midi_vel_offset:=ERR; { default return value }
  { only MIDI key events have velocity }
  offset:=midi_key_offset(eblk,chan);
  if (offset <> ERR) then
    midi_vel_offset:=offset+1;
  end;

{ Return MIDI velocity data from Track Event, if present
}
function get_midi_vel(eblk:track_event_block;
                     chan:byte):integer;
var
  vel_offset : integer;
begin
  get_midi_vel:=ERR; { default return value }
  vel_offset:=midi_vel_offset(eblk,chan);
  if (vel_offset <> ERR) then
    get_midi_vel:=eblk.event.mess[vel_offset];
  end;

{ Set MIDI velocity value in Track Event,
  if Track Event is of appropriate type
}
procedure set_midi_vel(var eblk:track_event_block;
                      chan,vel:integer);
var
  vel_offset : integer;
begin
  vel_offset:=midi_vel_offset(eblk,chan);
  if (vel_offset <> ERR) then
    eblk.event.mess[vel_offset]:=vel;
  end;

{ Scale all MIDI velocity data for
  a channel in a Track Data block
}
procedure scale_vel(var tdi,tdo:track_data_block;
                   chan:integer; vel_fact:real);
var
  curr_vel : integer;

```

(continued)

```

begin
reset_track_data(tdi);
reset_track_data(tdo);
while not(tdi.edat) do
begin
next_track_event(tdi);
curr_vel:=get_midi_vel(tdi.curr,chan);
if (curr_vel <> ERR) then
set_midi_vel(tdi.curr,chan,trunc(curr_vel*vel_fact));
store_track_event(tdo,tdi.curr);
end;
end;

```

```

====
LIST8.PAS
====

```

```

{ Redirect MIDI channel data in a
  Track Data Block to a new channel
}
procedure change_chan(var tdi,tdo:track_data_block;
                      old_chan,new_chan:byte);
begin
reset_track_data(tdi);
reset_track_data(tdo);
while not(tdi.edat) do
begin
next_track_event(tdi);
with tdi.curr do
begin
if (event_type = MIDI_RS) and
(midi_chan(running_status) = old_chan) then
event.mess[1]:=((event.mess[1] and $F0) or new_chan);
store_track_event(tdo,tdi.curr);
end;
end;
end;

{ Extract a single MIDI channel from a Track Data Block
}
procedure extract_chan(var tdi,tdo:track_data_block;
                      chan:byte);
begin
reset_track_data(tdi);
reset_track_data(tdo);
while not(tdi.edat) do
begin
next_track_event(tdi);
with tdi.curr do
begin
if (event_type in [MIDI_RS,MIDI])
and (midi_chan(running_status) <> chan) then
begin { convert to NOP }
event_type:=MARK;
event_len:=2;
event.mess[1]:=NOP;
end;
end;
store_track_event(tdo,tdi.curr);
end;
end;

{ Filter a MIDI channel from a Track Data Block
}
procedure filter_chan(var tdi,tdo:track_data_block; chan:byte);
begin
reset_track_data(tdi);
reset_track_data(tdo);
while not(tdi.edat) do
begin
next_track_event(tdi);
with tdi.curr do
begin
if (event_type in [MIDI_RS, MIDI])

```



```

    and (midi_chan(running_status) = chan) then
      begin { convert to NOP }
        event_type:=MARK;
        event_len:=2;
        event.mess[1]:=NOP;
      end;
    end;
    store_track_event(tdo,tdi.curr);
  end;
end;

```

```

=====
LIST9.PAS
=====

```

```

procedure quantize(var tdi,tdo:track_data_block;
                   quantum:integer);
var
  in_time : real;           { Actual elapsed time,
                             input track data block }
  out_time : real;          { Actual elapsed time,
                             output track data block }
  etime : integer;          { Temporary storage for
                             adjustment of event time }
  ground : integer;         { Rounding term }
begin
  reset_track_data(tdi);
  reset_track_data(tdo);
  ground:=quantum div 2;
  in_time:=0.0;
  out_time:=0.0;
  while not(tdi.edat) do
    begin
      next_track_event(tdi);
      with tdi.curr do
        begin
          etime:=event.time;
          { Adjust in/out time variance }
          etime:=etime - trunc(out_time-in_time);
          { quantize }
          etime:=trunc(quantum * ((etime + ground) div quantum));
          in_time:=in_time+event.time;
          out_time:=out_time+etime;
          event.time:=etime;
          while event.time > MAX_TIMING_COUNT do
            begin
              store_track_event(tdo,OVFL_EVENT);
              event.time:=event.time-MAX_TIMING_COUNT;
            end;
          store_track_event(tdo,tdi.curr);
        end;
      end;
    end;
  end;
end;

```

```

=====
LIST10.PAS
=====

```

```

var
  tdt1,tdt2 : track_data_block;

begin { main }
  load_track_data(tdt1);
  disp_track_data(tdt1);
  readln;

  writeln('Transposing pitch for channel 0');
  transpose_pitch(tdt1,tdt2,0,6);
  dump_track_data(tdt2);
  save_track_data(tdt2);

```

(continued)

```

writeln('extracting channel 0');
extract_chan(tdt1,tdt2,0);
disp_track_data(tdt2);
save_track_data(tdt2);

```

```

writeln('filtering channel 0');
filter_chan(tdt1,tdt2,0);
disp_track_data(tdt2);
save_track_data(tdt2);

```

```

end. { main }

```

```

list1.txt

```

```

TEXT
"Sorting ProDOS Directories," by Antonio C. Silvestri.
June, page 117. Also download list2.txt.

```

```

10  REM                PRODOS CATALOG SORT ROUTINE
20  REM                ANTONIO C. SILVESTRI
30  REM                SYSTEMS CONSULTANTS INC.
40  CLEAR: TEXT: HOME: DB = PEEK(115) + 256*PEEK(116)
50  FOR I=768 TO 792: READ H: POKE I,H: NEXT
60  DIM NA$(55), ST$(30), ST$(30), DL(10): N = 0: V = 2:
    V$ = "": GOSUB 460

70  VTAB 2: HTAB 6: PRINT "PRODOS FILENAME SORT UTILITY":
    VTAB 9: HTAB 10: PRINT "INSERT DISK IN ";:
    FLASH: PRINT "DRIVE 1";: NORMAL: PRINT:
    HTAB 9: PRINT "HIT ANY KEY TO CONTINUE"
80  POKE - 16368,0: WAIT - 16384,128: POKE - 16368,0

90  HOME: PRINT "SEARCHING FOR VALID FILENAMES": PRINT
100 IF N <= 0 THEN 450
110 GOSUB 480: BL = V: HE = V: HE$ = V$: CO = 0: BC = 0
120 POKE 791,BL - 256*INT(BL/256): POKE 792,INT(BL/256):
    POKE 776,128: CALL 768: IF PEEK(786) <> 0 THEN
        PRINT "ERROR IN READING BLOCK NO. ";BL: STOP
130 BC = BC + 1: DL(BC) = BL
140 FOR J=0 TO 12: IF J <> 0 THEN 180
150 IF BL=HE THEN DR$ = "":
    FOR I=0 TO 38: DR$ = DR$ + CHR$(PEEK(DB + 4 + I)): NEXT:
    HE$ = HE$ + "/" + MID$(DR$,2,PEEK(DB+4)-16*INT(PEEK(DB+4)/16)):
    PRINT "READING DIRECTORY: ";HE$
160 PRINT "BLOCK NO. ";BL;" READ"
170 IF BL = HE THEN 210
180 IF PEEK (DB + 4 + J * 39) = 0 THEN PRINT "D";: GOTO 210
190 PRINT ".": CO = CO + 1: NA$(CO) = "":
    FOR I=0 TO 38: NA$(CO)=NA$(CO)+CHR$(PEEK(DB+4+J*39+I)): NEXT
200 IF ASC(MID$(NA$(CO),17,1))=15 THEN
    V = ASC(MID$(NA$(CO),18,1)) + 256*ASC(MID$(NA$(CO),19,1)):
    V$ = HE$: GOSUB 460
210 NEXT J: PRINT: PRINT: X = FRE (0):
    BL = PEEK(DB+2) + 256*PEEK(DB+3): IF BL <> 0 THEN 120

220 IF CO=0 THEN HOME: VTAB 10: FLASH:
    PRINT "***WARNING**"; CHR$ (7):: NORMAL:
    PRINT "NO FILENAMES CAN BE FOUND": GOTO 290
230 IF CO=1 THEN 290
240 HOME: VTAB 10: PRINT CO;" FILENAMES FOUND";: HTAB 29:
    PRINT "NOW ";: FLASH: PRINT "SORTING": NORMAL: NX = CO
250 VTAB 17: HTAB 8: PRINT "FILENAMES HAVE BEEN PLACED"
260 VTAB 17: HTAB 7 - LEN(STR$(CO - NX)): PRINT CO - NX
270 FLAG = 0: FOR I = 2 TO NX:
    IF MID$(NA$(I),2,15) < MID$(NA$(I-1),2,15) THEN
        H$ = NA$(I): NA$(I) = NA$(I-1): NA$(I - 1) = H$:
        FLAG = 1
280 NEXT: X = FRE(0): NX = NX - 1:
    IF FLAG = 1 AND NX <> 1 THEN 260

```



```

290 HOME: PRINT "STORING PURGED AND SORTED DIRECTORY": PRINT
300 A = 1: B = 0: FOR J = 1 TO BC
310 PRINT "NOW FORMING BLOCK NO. "; DL(J)
320 IF J = 1 THEN AX = 0: GOTO 340
330 AX = DL(J - 1)
340 IF J = BC THEN BX = 0: GOTO 360
350 BX = DL(J + 1)
360 POKE DB,AX-256*INT(AX/256): POKE DB+1, INT(AX/256):
POKE DB+2,BX-256*INT(BX/256): POKE DB+3, INT(BX/256):
POKE DB+511,0
370 IF J = 1 THEN
FOR K=1 TO 39: POKE DB+3+B*39+K,ASC(MID$(DR$,K,1)): NEXT:
PRINT ".":B = B + 1
380 IF A <= CO THEN
FOR K=1 TO 39: POKE DB+3+B*39+K,ASC(MID$(NA$(A),K,1)): NEXT:
PRINT ".":GOTO 400
390 FOR K=1 TO 39: POKE DB+3+B*39+K,0: NEXT: PRINT "Z";
400 A = A+1: B = B+1: IF B < 13 THEN 380
410 PRINT: PRINT: B = 0: POKE 791,DL(J)-256*INT(DL(J)/256):
POKE 792, INT(DL(J)/256): POKE 776,129: CALL 768
420 IF PEEK(786) <> 0 THEN
PRINT "ERROR IN WRITING BLOCK NO. ";DL(J): STOP
430 NEXT J: GOTO 90
440 DATA 169, 0, 141, 18, 3, 32, 0, 191, 0, 19, 3, 176, 1,
96, 238, 18, 3, 96, 0, 3, 96, 0, 150, 0, 0
450 END
460 IF N >= 30 THEN
PRINT "STACK OVERFLOW": STOP
470 N = N+1: ST(N)=V: ST$(N)=V$: RETURN
480 IF N <= 0 THEN
PRINT "STACK UNDERFLOW": STOP
490 V = ST(N): V$ = ST$(N): N = N-1: RETURN

```

The PRODOS Diskette Sorting Utility
Listing #1

(continued)

list2.txt

TEXT

"Sorting ProDOS Directories," by Antonio C. Silvestri.
June, page 117. Also download list1.txt.

```

10 DATA "/SCRATCH", "/SCRATCH/MASS", "/SCRATCH/RHODE",
    "/SCRATCH/VERMONT", "/SCRATCH/MASS/NY",
    "/SCRATCH/MASS/NJ", "/SCRATCH/RHODE/PA",
    "/SCRATCH/VERMONT/MAINE"
20 DIM A$(40): D$=CHR$(4): ONERR GOTO 100
30 READ H$: PRINT D$;"PREFIX ";H$: PRINT D$;"SAVE TESTFILE"
40 J=0: L=INT(40*RND(1))+1: PRINT: PRINT:
    PRINT L;" FILES TO BE CREATED": PRINT
50 FOR K=1 TO L: TY$="": FOR I=1 TO INT(10*RND(1))+1:
    TY$=TY$+CHR$(65+26*RND(1)): NEXT: X=FRE (0)
60 PRINT K;" ";H$+ "/" +TY$: PRINT D$;"OPEN "+TY$:
    PRINT D$;"CLOSE "+TY$: IF RND(1) < .30 THEN J=J+1: A$(J)=TY$
70 NEXT K
80 PRINT J;" FILES TO DELETE": IF J=0 THEN 30
90 FOR I=1 TO J: PRINT D$;"DELETE "+A$(I):
    PRINT I;" ";H$+ "/" +A$(I): NEXT: GOTO 30
100 END

```

A Program to Create a Testing Diskette for
the ProDOS Sorting Utility

Listing #2

indexbpp.pas

TEXT

Programming Project: "A Simple File-Indexing System," by Bruce Webster.
June, page 92. Also download indexbpp.lst.

{\$V-}

program FileIndex;

const

```

    IndexMax      = 1000;
    RecCountErr   = -2;
    NewFileCreated = -1;
    NoError       = 0;
    RecordNotFound = 1;
    NoMoreRoom    = 2;
    AlreadyExists  = 3;
    OutOfRange    = 4;

```

type

```

    Keytype      = string[40];
    FileStr      = string[80];
    Whatever     = string[12];

```

DataRec = record

case Boolean of

```

    True      : (NumRecs   : Integer);
    False     : (Key       : Keytype;
                 theRest   : Whatever);

```

end;

IndexRec = record

```

    Key      : Keytype;
    Num      : Integer;
end;

```



```

IndexList    = array[1..IndexMax] of IndexRec;

var
  KList      : IndexList;
  DFile      : file of DataRec;
  MaxRec     : Integer;

{
  compiler-specific file I/O routines
  these procedures are specific to TURBO Pascal.  If you
  are using another Pascal compiler, you will need to
  modify them appropriately.  Note that TURBO Pascal does
  not support the standard routines GET and PUT, but instead
  uses READ and WRITE.
}

{$I-} { turn off I/O error checking }

procedure FRead(RNum : Integer; var Rec : DataRec; var Error : Integer);
{
  reads record #RNum into Rec
}
begin
  if (RNum < 0) or (RNum > MaxRec)
  then Error := OutOfRange
  else begin
    Seek(DFile,RNum);
    Read(DFile,Rec);
    Error := IOResult;
    if Error > 0
    then Error := 100 + Error
  end
end; { of proc FRead }

procedure FWrite(RNum : Integer; Rec : DataRec; var Error : Integer);
{
  writes record #RNum into Rec
}
begin
  if (RNum < 0) or (RNum > MaxRec)
  then Error := OutOfRange
  else begin
    Seek(DFile,RNum);
    Write(DFile,Rec);
    Error := IOResult;
    if Error > 0
    then Error := 100 + Error
  end
end; { of proc FRead }

procedure FOpen(FileName : FileStr; var Error : Integer);
{
  tries to open FileName; if it doesn't exist, creates
  it with the appropriate header record
}
const
  TurboNoFile = 1; { "no file" error code for TURBO Pascal }
  NoIOError   = 0;
var
  IOCode      : Integer;
  TRec        : DataRec;
begin
  Assign(DFile,FileName);
  Reset(DFile);
  IOCode := IOResult;
  if IOCode = TurboNoFile then begin { file doesn't exist }
    FillChar(TRec,SizeOf(TRec),0);
    Rewrite(DFile);
    TRec.NumRecs := 0;
    Write(DFile,TRec);
    Close(DFile);
    Assign(DFile,Filename);
    Reset(DFile);
    IOCode := IOResult;
    if IOCode = NoIOError
    then Error := NewFileCreated
  end;
  if IOCode <> NoIOError

```

(continued)

```

    then Error := 100 + IOCode;
end; { of proc FOpen }

procedure FClose(var Error : Integer);
{
    closes file
}
begin
    Close(DFile);
    Error := IOResult;
    if Error > 0
    then Error := Error + 100
end; { of proc FClose }

{$I+} { turn on I/O error checking }

{
    initialization and cleanup routines
}

procedure SortIndexList;
{
    sorts the array KList using a selection sort technique
}
var
    I,J,Min      : Integer;
    Temp         : IndexRec;
begin
    for I := 1 to MaxRec-1 do begin
        Min := I;
        for J := I+1 to MaxRec do
            if KList[J].Key < KList[Min].Key
            then Min := J;
        Temp := KList[I];
        KList[I] := KList[Min];
        KList[Min] := Temp
    end
end; { of proc SortIndexList }

procedure InitStuff(FileName : FileStr; var Error : Integer);
{
    sets everything up for indexing system. This assumes that
    there are no more than IndexMax (=1000) records, and that the
    records are numbered 1..IndexMax. Record #0 is the header
    record and is used to store the current number of records
    actively being used in the file
}
var
    Indx,TErr      : Integer;
    TRec           : DataRec;
begin
    Error := NoError;
    FOpen(FileName,Error);
    if Error <= NoError then begin
        MaxRec := 0;
        FRead(0,TRec,TErr);
        Error := TErr;
        MaxRec := TRec.NumRecs;
        for Indx := 1 to MaxRec do begin
            FRead(Indx,TRec,TErr);
            if TErr > 0
            then Error := TErr;
            KList[Indx].Key := TRec.Key;
            KList[Indx].Num := Indx
        end;
        SortIndexList
    end
end; { of proc InitStuff }

procedure CleanUpStuff(var Error : Integer);
{
    this just does an orderly shutdown and should be called
    before you leave your program (or open another data file)
}
var
    TRec           : DataRec;
begin
    TRec.NumRecs := MaxRec; { save out # of records }

```



```

    FWrite(0,TRec,Error);
    FClose(Error)
end; { of proc CleanUpStuff }

function FindKey(Key : Keytype) : Integer;
{
    looks for Key in KList; returns location in KList
    if found; otherwise returns - 1
}
var
    L,R,Mid      : Integer;
begin
    L := 1; R := MaxRec;
    repeat
        Mid := (L+R) div 2;
        if Key < KList[Mid].Key
            then R := Mid-1
            else L := Mid+1
    until (Key = KList[Mid].Key) or (L > R);
    if Key = KList[Mid].Key
        then FindKey := Mid
        else FindKey := -1
end; { of proc FindKey }

procedure GetRecord(Key : Keytype; var Rec : DataRec;
    var Error : Integer);
{
    looks through KList for Key; if found, returns in Rec.
    It and the routines that follow assume the procedure Seek
    for random access of the file of records.
}
var
    Item          : Integer;
begin
    Error := NoError;
    Item := FindKey(Key);
    if Item > 0
        then FRead(KList[Item].Num,Rec,Error)
        else Error := RecordNotFound
end; { of proc GetRecord }

procedure PutRecord(Rec : DataRec; var Error : Integer);
{
    writes Rec out to the file. If a record with that
    key already exists, then overwrites that record;
    otherwise, adds the record to the end of the file.
    If there's no more room for records, exits with an
    error code
}
var
    Item          : Integer;
begin
    Error := NoError;
    Item := FindKey(Rec.Key);
    if Item >= 0
        then FWrite(KList[Item].Num,Rec,Error)
        else if MaxRec < IndexMax then begin
            MaxRec := MaxRec + 1;
            FWrite(MaxRec,Rec,Error);
            KList[MaxRec].Key := Rec.Key;
            KList[MaxRec].Num := MaxRec;
            SortIndexList
        end
        else Error := NoMoreRoom
end; { of proc PutRecord }

procedure AddRecord(Rec : DataRec; var Error : Integer);
{
    adds a record to the file. If a record with the same
    key already exists, then exits with an error code
}
var
    Item          : Integer;
begin
    Error := NoError;

```

(continued)

```

Item := FindKey(Rec.Key);
if Item > 0
  then Error := AlreadyExists
  else PutRecord(Rec,Error)
end; { of proc AddRecord }

procedure DeleteRecord(Key : Keytype; var Error : Integer);
{
  deletes the record with 'Key' by copying the last record
  in the file to that slot, then modifies KList by shuffling
  all the key entries up
}
var
  Item,Last,Max,MVal      : Integer;
  TRec                   : DataRec;
begin
  Error := NoError;
  Item := FindKey(Key);
  if Item = -1
    then Error := RecordNotFound
  else begin
    Max := 1; MVal := KList[Max].Num;
    for Last := 2 to MaxRec do
      if KList[Last].Num > MVal then begin
        Max := Last; MVal := KList[Last].Num
      end;
    if Max <> Item then begin
      FRead(MVal,TRec,Error); { get last record in file }
      FWrite(KList[Item].Num,TRec,Error); { write over it }
      KList[Max].Num := KList[Item].Num
    end;
    for Last := Item to MaxRec-1 do { delete KList[Item] }
      KList[Last] := KList[Last+1];
    MaxRec := MaxRec - 1 { adjust # of records }
  end
end; { of proc DeleteRecord }

{
  USERIO.LIB

  procedure and functions in this library

  WriteStr      write message out at (Col,Line)
  Error         writes message out at (1,1), waits for character
  GetChar       prompt user for one of a set of characters
  Yes           gets Y/N answer from user
  GetString     prompt user for a string
  IOCheck       checks for I/O error; prints message if necessary
}

type
  MsgStr      = string[80];
  CharSet     = set of Char;

var
  IOErr       : Boolean;
  IOCode      : Integer;

procedure WriteStr(Col,Line : Integer; TStr : MsgStr);
{
  purpose      writes message out at spot indicated
  last update  23 Jun 85
}
begin
  GoToXY(Col,Line); ClrEol;
  Write(TStr)
end; { of proc WriteStr }

procedure Error(Msg : MsgStr);
{
  purpose      writes error message out at (1,1); waits for character
  last update  05 Jul 85
}
const

```



```

    Bell                = ^G;
var
    Ch                  : Char;
begin
    WriteStr(1,1,Msg+Bell+' (hit any key) ');
    Read(Kbd,Ch)
end; { of proc Error }

procedure GetChar(var Ch : Char; Prompt : MsgStr; OKSet : CharSet);
{
    purpose             let user enter command
    last update         23 Jun 85
}
begin
    WriteStr(1,1,Prompt);
    repeat
        Read(Kbd,Ch);
        Ch := UpCase(Ch)
    until Ch in OKSet;
    WriteLn(Ch)
end; { of proc GetChar }

function Yes(Question : MsgStr) : Boolean;
{
    purpose             asks user Y/N question
    last update         03 Jul 85
}
var
    Ch                  : Char;
begin
    GetChar(Ch,Question+' (Y/N) ',['Y','N']);
    Yes := (Ch = 'Y')
end; { of func Yes }

procedure GetString(var NStr : MsgStr; Prompt : MsgStr; MaxLen : Integer;
                    OKSet : CharSet);
{
    purpose             get string from user
    last update         09 Jul 85
}
const
    BS                  = ^H;
    CR                  = ^M;
    ConSet              : CharSet = [BS,CR];
var
    TStr                : MsgStr;
    TLen,X              : Integer;
    Ch                  : Char;
begin
    {$I-} { turn off I/O checking }
    TStr := '';
    TLen := 0;
    WriteStr(1,1,Prompt);
    X := 1 + Length(Prompt);
    OKSet := OKSet + ConSet;
    repeat
        GoToXY(X,1);
        repeat
            Read(Kbd,Ch)
        until Ch in OKSet;
        if Ch = BS then begin
            if TLen > 0 then begin
                TLen := TLen - 1;
                X := X - 1;
                GoToXY(X,1); Write(' ');
            end
        end
        else if (Ch <> CR) and (TLen < MaxLen) then begin
            Write(Ch);
            TLen := TLen + 1;
            TStr[TLen] := Ch;
            X := X + 1;
        end
    until Ch = CR;

```

(continued)

```

if TLen > 0 then begin
  TStr[0] := Chr(TLen);
  NStr := TStr
end
else Write(NStr)
{$I+}
end; { of proc GetString }

procedure IOCheck(IOCode : Integer);
{
  purpose      check for IO error; print message if needed
  last update   19 Feb 86
}
var
  TStr          : string[4];
begin
  IOErr := (IOCode <> 0);
  if IOErr then case IOCode of
    $01 : Error('IOERROR> File does not exist');
    $02 : Error('IOERROR> File not open for input');
    $03 : Error('IOERROR> File not open for output');
    $04 : Error('IOERROR> File not open');
    $10 : Error('IOERROR> Error in numeric format');
    $20 : Error('IOERROR> Operation not allowed on logical device');
    $21 : Error('IOERROR> Not allowed in direct mode');
    $22 : Error('IOERROR> Assign to standard files not allowed');
    $90 : Error('IOERROR> Record length mismatch');
    $91 : Error('IOERROR> Seek beyond end of file');
    $99 : Error('IOERROR> Unexpected end of file');
    $F0 : Error('IOERROR> Disk write error');
    $F1 : Error('IOERROR> Directory is full');
    $F2 : Error('IOERROR> File size overflow');
    $FF : Error('IOERROR> File disappeared');
  else Str(IOCode:3,TStr);
        Error('IOERROR> Unknown I/O error: '+TStr)
  end
end; { of proc IOCheck }

{
  declarations and code for test program
}
const
  CmdPrompt      : MsgStr =
    'TEST> A)dd, D)delete, F)ind, L)ist, I)ndex, C)lose, Q)uit: ';
  FilePrompt     : MsgStr = 'TEST> Enter file name: ';
  DonePrompt     : MsgStr = 'TEST> Another file?';

  CmdSet         : CharSet = ['A','D','F','L','I','C','Q'];
  NameSet        : CharSet = [' ','..','~'];
  PhoneSet       : CharSet = ['0'..'9','-','/','(',')'];

var
  Cmd            : Char;
  ErrVal         : Integer;
  FileName       : FileStr;
  Done           : Boolean;

procedure FileError(ErrVal : Integer);
begin
  if ErrVal < 100 then case ErrVal of
    RecCountErr   : Error('Record count mismatch');
    NewFileCreated : Error('Creating new file');
    RecordNotFound : Error('Record not found');
    NoMoreRoom     : Error('No more room');
    AlreadyExists  : Error('Record already exists')
  end
  else begin
    IOCheck(ErrVal-100)
  end
end; { of proc FileError }

```



```

procedure DoAdd;
{
    purpose      add a record to the file
    last update  19 Feb 86
}
var
    TStr      : MsgStr;
    TRec      : DataRec;
begin
    FillChar(TRec,SizeOf(TRec),0);
    with TRec do begin
        TStr := '';
        GetString(TStr,'ADD> Enter name: ',40,NameSet);
        if TStr <> '' then begin
            Key := TStr; TStr := '';
            GetString(TStr,'ADD> Enter phone #: ',12,PhoneSet);
            theRest := TStr;
            AddRecord(TRec,ErrVal);
            Flush(DFile);
            FileError(ErrVal)
        end
    end;
end; { of proc DoAdd }

procedure DoDelete;
{
    purpose      delete a record from the file
    last update  19 Feb 86
}
var
    Key      : Keytype;
begin
    GetString(Key,'DELETE> Enter name: ',40,NameSet);
    DeleteRecord(Key,ErrVal);
    FileError(ErrVal)
end; { of proc DoDelete }

procedure DoFind;
{
    purpose      find a record in the file
    last update  19 Feb 86
}
var
    Key      : Keytype;
    TRec      : DataRec;
begin
    GetString(Key,'FIND> Enter name: ',40,NameSet);
    GetRecord(Key,TRec,ErrVal);
    if ErrVal = NoError then begin
        WriteStr(1,2,'The phone number is ');
        WriteLn(TRec.theRest)
    end
    else FileError(ErrVal)
end; { of proc DoDelete }

procedure DoList;
{
    purpose      list out contents of the file
    last update  19 Feb 86
}
var
    TRec      : DataRec;
    Indx      : Integer;
begin
    ClrScr; WriteLn;
    for Indx := 1 to MaxRec do with KList[Indx] do begin
        WriteStr(1,Indx+1,Key); Write(' ': (45-Length(Key)));
        GetRecord(Key,TRec,ErrVal);
        if ErrVal = NoError then with TRec do
            WriteLn(theRest)
        else FileError(ErrVal)
    end
end; { of proc DoList }

```

(continued)

June

```
procedure ShowIndex;
{
    purpose      list out contents of the key list
    last update  19 Feb 86
}
var
    Indx          : Integer;
begin
    ClrScr; Writeln;
    for Indx := 1 to MaxRec do with KList[Indx] do
        Writeln(Key, ' ':(45-Length(Key)),Num:5)
    end; { of proc DoList }

begin
    repeat
        Done := False;
        ClrScr;
        GetString(FileName,FilePrompt,80,NameSet);
        InitStuff(FileName,ErrVal);
        FileError(ErrVal);
        repeat
            GetChar(Cmd,CmdPrompt,CmdSet);
            case Cmd of
                'A' : DoAdd;
                'D' : DoDelete;
                'F' : DoFind;
                'L' : DoList;
                'I' : ShowIndex;
                'Q' : Done := True;
            end
        until (Cmd = 'C') or Done;
        CleanUpStuff(ErrVal);
        FileError(ErrVal);
        ClrScr;
        if not Done
            then Done := not Yes(DonePrompt)
        until Done
    end. { of program TestIndex }
```

mid111.c

TEXT
"A MIDI Project," by Jay Kubicky.
June, page 199. Also download rxint11.a.

```
/* MIDInterface 1.11
   Copyright (C) 1985, 1986 By:
       Jay Kubicky
       934 N. Orange St.
       Media, PA 19063
       215-565-7761
```

This is a VERY recent version of this program.
It has not been 'tested' extensively, though all testing that
has been done shows favorable results.
All users are free, of course, to test (and debug) this software as
desired.

This program was developed under the DeSmet C compiler, and utilizes DeSmet's vastly useful in-line assembly language capability. Compiling it on other compilers (such as Lattice) may require EXTENSIVE modification.

2/14/86

```

*/
#define MIDID 0xffa0
#define MIDIS 0xffa2
#define PIC0 0x20
#define PIC1 0x21
#define CSTAT 0xFFA7
#define COUNTER1 0xFFA4
#define COUNTER2 0xFFA5
#define Tsize 25600 /* track size in bytes (24K) */
#define Tk 25 /* track size in K */
#define seg_per_trk 1600 /* track size in paragraphs */
/* the following structures hold data for each buffer */

char prog_id[]={"MIDIInterface 1.11 (c) 1985, 1986 Jay Kubicky"};

struct rec { /* 63 bytes long 63*16=1008 bytes total */
    char chan;
    unsigned ssize;
    char name[40];
    char transpose; /* transpose amount */
    char extra[19]; /* for later use */
    } parts[16];

char in_filt; /* This is the MIDI input filter mask:
               bit: message filtered out:
               ----
               0 Note ON
               1 Note OFF
               2 Program Change
               3 Channel Pres. (After-touch)
               4 Pitch Wheel
               5 Control Change
               6 Poly. Key pres. (After-touch)
               */
char on[4]="ON "; /* 'ON' */
char off[4]="OFF"; /* 'OFF' */
char yes[4]="YES";
char no[4]="NO ";
char sin_line[80]; /* A 79 char. single line */
unsigned r_segment;
char *ptr, *end;
int fd; char filename[30];

int r_tempo; /* tempo val (bpm) */
unsigned tempo; /* real tempo val. */
char destbyte, stop;
char clsb, cmsb, dsync, MIDIsync, audmet;
char rbuff; /* receive buffer */
char beat, mbeat, abeat, s_t_b, m_t_b, a_t_b; /* beats & time bases */

char buffon[16]; /* true means given buff is active */
unsigned pointers[16]; /* an array of pointers into present buffers */
unsigned startp[16]; /* an array of starts of present buffer */
unsigned endp[16]; /* pointers to end of songs */
unsigned segment[16]; /* array of segments of tracks */

int pfd; /* fd for printer */
char printer; /* printer enable */

extern unsigned _rax, _rbx, _rdx, _res, _rds;

```

(continued)

```

char LAST_STAT;
char C_O_F;
char clk_type;          /* 0=internal, 1=external */
char counter_dec;
char vir_buff[1024];
char buff1[Tsize];      /* track 0 - 3070 notes */
char buff2[Tsize];      /* track 1 - 3070 notes */
/* first two tracks eat up 50K */
/* last fourteen will take 350K combined */
/* total note storage=
    3200 notes per track
    51200 notes in all!!!! */

char *_showsp(),*_showss(),*_showds();
main()
{
    unsigned a;
    char clkval,sel,b;    int c;
    unsigned base_seg, cont_mem;

    /* the following code stores the DS at 0x4FA */
    /* THIS IS VERY IMPORTANT!!!!!! */
    /* this allows the DS to be transferred to the RXINT function */

```

```

# asm
MOV CX,DS ; store DS in CX
PUSH DS ; save DS
MOV AX,0 ; this will be new value for DS
MOV DS,AX ; put 0 in DS
MOV WORD [04FAH], CX ; store DS
POP DS ; retrieve DS
#

```

/* MIDInterface 1.11 16-track digital sequencer

Data storage format:

(MIDI ORIENTED COMMANDS)					
byte num:	first	second	third	fourth	fifth
Note ON	00vvvvvv	LLLLLLLL	MMMMMMMM	1nnnnnnn	--
Note OFF	00vvvvvv	LLLLLLLL	MMMMMMMM	0nnnnnnn	--
Prog Chan	01000000	LLLLLLLL	MMMMMMMM	0ppppppp	--
Chan Pres (after-touch)	01000001	LLLLLLLL	MMMMMMMM	0aaaaaaa	--
Pitch Wheel	10wwwww	LLLLLLLL	MMMMMMMM	--	--
Cntrl Chan	11000000	LLLLLLLL	MMMMMMMM	0nnnnnnn	0ccccccc
Key Pres. (after-touch)	011vvvvv	LLLLLLLL	MMMMMMMM	0nnnnnnn	--

(INTERNAL ORIENTED COMMANDS)					
Change Tempo	11000001	LLLLLLLL	MMMMMMMM	tttttttt	--
End	11111111	--	--	--	--

vvvvvv - top 6 bits of velocity (NOTE ON or OFF)
LLLLLLLL - LSB of clk
MMMMMMMM - MSB of clk
nnnnnnn - note for either NOTE ON or NOTE OFF
ppppppp - new program
aaaaaaa - chan pressure (after-touch)
wwwww - top six of bender
nnnnnnn - control number
ccccccc - control value

tttttttt - new tempo value (40-200)

*/

c=0;


```

while(c<79)
    sin_line[c]=sin_line[c+1];
sin_line[c]='\0';

in_filt=0xff; /* Don't filter out anything */
clk_type=0; /* Internal */
counter_dec=2;
rbuff=255;
segment[0]=segment[1]=_showseg(); /* 1st two buffs in DS */
startp[0]=buff1; endp[0]=buff1+Tsize-8;
startp[1]=buff2; endp[1]=buff2+Tsize-8;
parts[0].ssize=0; parts[1].ssize=0;

/* This is the memory allocation section.
   This is really a sort of pseudo-allocation scheme
   allowing the use of the entire system memory (up to 640K).
   Memory is allocated in blocks of 25k (1600 paragraphs).
   All tracks are allocated sequentially until you run out
   of system RAM. The first two tracks are taken care of
   in the C memory, so there should always be room for these.
*/

base_seg=_showseg(); /* start just after stack */
base_seg += (_showseg())/16; /* This is the segment boundary of
                               the bottom of un-initialized mem */
++base_seg; /* one for a safety */

cont_mem=get_ov_mem();
cont_mem += 64; /* Find total # of system para. */

/* Here's where we "allocate our memory" */

_setmem(segment+4,25,0);
c=2;
for (a=base_seg; a < cont_mem && c < 16; a+=seg_per_trk) {
    segment[c]=a;
    startp[c]=a;
    endp[c]=a+Tsize-8;
    parts[c].ssize=0;
    parts[c].transp=0;
    ++c;
}

m_t_b=2; s_t_b=1; a_t_b=45; /* default time bases */
printer=0;
scr_cla();
printf("Do you wish to use the printer as an audio metername?");
if(toupper(getchar())=='Y') {
    printf("Already the printer and hit any key");
    getchar();
    if((pfd=open("PRN",2))!=-1) {
        printf("Error opening printer ... aborting\n");
        getchar();
        exit(1);
    }
    bleep();
    printer=1;
}

start:
    setdart();
    intoff(); /* turn interrupts off */
    scr_cla();

printf("
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n");
printf("
? MIDIInterface 3.11 16 Track Digital Recorder V\n");
printf("
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n");
printf("\n");
printf("
Main Menu:\n");
printf("
0000000000000000\n");
printf("
1.....Erase a track\n");
printf("
2.....Record to a track\n");
printf("
3.....Play from a track\n");
printf("
4.....Track information\n");
printf("
5.....Save a track\n");
printf("
6.....Load a track\n");
printf("
7.....Set MIDI modes\n");
printf("
8.....Edit input filter\n");
printf("
9.....Quit\n");

```

(continued)

```

/*
DBUFFS
This is the general buffer display routine.
It displays all pertinent data for all available buffers.
*/
dbuffs()
{
    char a,b; unsigned tn,nf,nu;
    scr_cla();
    printf("                                Track Assignment Screen:\n");
    printf("                                vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv\n");
    printf("Track Num:   Transmit Chan:   Notes Used:   Transpose:   Enabeled:\n");
    printf("%s\n",sin_line);
    for(a=0; a<16; ++a) {
        if(segment[a]) {
            tn=(endp[a]-startp[a])/8;
            nf=tn-(parts[a].ssize/8);
            nu=tn-nf;
            b=parts[a].transpose;
            printf("%d\t\t%d\t\t%d\t\t%d\t\t%s\n",a,parts[a].chan,nu,
                b < 128 ? b : -256+b, buffon[a]
            );
        }
        else
    }
}

```



```

        printf("--\t\t--\t\t--\t\t--\t\t--\n");
    }
    printf("%s\n",sin_line);
ttag: scr_rowcol(21,0);
    printf("Change one trans. channel (1), change all (2), transpose(3),
enable status (4)\n");
    printf("or (0) to quit/continue: ");
    a=getchar();
    scr_rowcol(22,17);
    scr_cls(); /* clear rest of current line */
    switch(a)
    {
        case '0':
            return;
            break;
        case '1':
            printf("    Enter track number:");
            b=getint();
            scr_rowcol(23,0);
            printf("    Enter new transmit channel:");
            parts[b].chan=getint();
            break;
        case '2':
            printf("    Enter new transmit channel for all tracks:");
            a=getint();
            for(b=0; b<16; ++b)
                parts[b].chan=a;
            break;
        case '3':
            printf("    Enter track number:");
            b=getint();
            scr_rowcol(23,0);
            printf("    Enter new transpose amount:");
            parts[b].transpose=getint();
            break;
        case '4':
            printf("    Enter track number:");
            b=getint();
            if(parts[b].ssize && b != rbuff)
                buffon[b]= buffon[b] ? 0 : 1;
            break;
        default:
            goto ttag;
    }
    goto s_;
}

/*      ERBUFF
This funct. erase a buffer (set the size to 0)
*/
erbuff()
{
    scr_cla();
    char b;
    printf("                                Erase a Track\n");
    printf("%s\n",sin_line);
    printf("\nWhich buffer? ");
    b=getint();
    printf("Are you sure you want to erase buffer %d?",b);
    if(toupper(getchar()) != 'Y')
        return;
    parts[b].ssize=0;
    buffon[b]=0;
}

/*      PBUFFS
This is the top-level play-buffer routine.
*/
pbufs()
{
    int m;
    copyptrs();

    scr_cla();
    printf("                                Play Track Mode\n");
    printf("%s\n\n",sin_line);

```

(continued)

```

printf("    Hit any key to continue\n");
getchar();
dbuffs();

scr_cla();
printf("                                Play Track Mode\n");
printf("%s\n\n",sin_line);

get_options();

printf("\nHit space bar to continue, any other key to quit\n");
if(getchar()!=32)
    return;
setrec();
destbyte=0xff;
play();
intoff();
_outb(5,MIDIS);    /* send off to drum mach. */
_outb(104,MIDIS);
_outb(0xfc,MIDID); /* MIDI seq off */
for(m=0; m<16; ++m)
    if(parts[m].ssize)
        buffon[m]=1;
}

/*
RECBUFF:
This is the track record routine.
All it really does is set up & enable the interrupt routine -
RxInt
*/
recbuff()
{
    char b; int m;
    copyptrs();
    scr_cla();
    printf("                                Record a Track\n");
    printf("%s\n",sin_line);
    printf("    Which buffer:");
    b=getint();
    if(parts[b].ssize)
    {
        printf("That buffer already has music, still want to record (Y/N)?");
        if(toupper(getchar()) != 'Y')
            return;
    }

    if(!segment[b])
    {
        printf("Sorry, that track is not available to record in.\n");
        hak();
        return;
    }

    rbuff=b;
    parts[b].ssize=0;
    dbuffs(); /* display buff enable board w/ option to change */
    scr_cla();

    get_options();

    printf("\nHit any key to begin recording\n");
    getchar();
    scr_cla();
    printf("Hit any key to stop recording\n");

    setrec();
    play();

    _poke(255,ptr,segment[rbuff]); /* EOS (end-of-song) */
    parts[b].ssize=ptr-startp[rbuff];
    printf("\nSong is %d notes long\n",parts[b].ssize / 8);
    rbuff=255; /* no more record buffer */
    pointers[b]=startp[b]; /* copy start pointer */
    buffon[b]=1; /* enable present buffer */

    _outb(5,MIDIS); /* send off to drum mach. */
    _outb(104,MIDIS);
    _outb(0xfc,MIDID); /* MIDI seq off */

```



```

    rbuff=0xff;
    hak();
}

/*      SETREC
    This sets up almost everything for Play & Record.
*/
setrec()
{
    int a;

    stop=0;
    destbyte=0;
    setint();
    ptr=startp[rbuff];
    end=endp[rbuff];
    r_segment=segment[rbuff];
    end=endp[rbuff];
    C_O_F=0x90;
    beat=abeat=abect=0;
    if(dsinc)
        _outb(5,ACCTS); /* send start to drums */
        _outb(10,ACCTS);
    }
    if(MIDIsync)
        _outb(0,ACCTS); /* start ext seq */
    pointers[0]=startp[0];
    pointers[1]=startp[1];
    setcounter(tema & Buff,tema >> 8,255,255);
}

copyptrs() /* copy start points of buffers into pointer areas */
{
    char a;
    for(a=0; a<15; ++a)
        pointers[a]=startp[a];
}

/*      LTRACK
    Load a track into memory.
*/
ltrack()
{
    char tr;
    scr_clea();
    printf("                                Load a Track\n");
    printf("%s\n",sin_line);
    dir("???????", "???");
    scr_rowcol(2,0);
    printf("    Load which track:");
    tr=getint();
    if(parts[tr].ssize)
    {
        printf("Data already in that track ... aborting.\n");
        getchar(); return;
    }
    printf("    Load from which file:");
    scanf("%s",filename);
    if((fd=open(filename,2)) == -1) {
        printf("Can't open %s ... aborting\n",filename);
        getchar();
        return;
    }
    if(read(fd,parts[tr],63) == -1) {
        printf("Error encountered during reading.\n");
        getchar();
        return;
    }
    if(seg_read(fd,tr,parts[tr].ssize+1) == -1) {
        printf("Error encountered while reading.\n");
        getchar(); return;
    }
    close(fd);
    printf("Total bytes loaded from disk:%d\n",parts[tr].ssize+63);
    buffon[tr]=1;
    getchar();
}

```

(continued)

```

/*      STRACK
      Save a track to disk.
*/
strack() /* save a track */
{
    char tr;
    scr_clea();
    printf("                                Save a Track\n");
    printf("%s\n",sin_line);
    dir("???????", "???");
    scr_rowcol(2,0);
    printf("    Save which track:");
    tr=getint();
    if(!parts[tr].ssize)
    {
        printf("Nothing in that track ... aborting.\n");
        getchar(); return;
    }
    printf("    Save to what file:");
    scanf("%s",filename);
    if((fd=creat(filename)) == -1) {
        printf("Can't creat %s ... aborting\n",filename);
        getchar();
        return;
    }
    if(write(fd,parts[tr],63) == -1) {
        printf("Error encountered during writing.\n");
        getchar();
        goto _end;
    }
    if(seg_write(fd,tr,parts[tr].ssize+1) == -1) {
        printf("Error encountered while writing.\n");
        getchar(); goto _end;
    }
    close(fd);
    printf("Total bytes written to disk:%d\n",parts[tr].ssize+63);
    hak();
    return;
_end:
    close(fd);
}

/*      SEG_READ, SEG_WRITE
      These functions carry out Intra-segmentary disk I/O functions.
*/
seg_read(fd,tr,size)
int fd, tr; unsigned size;
{
    unsigned a;
    a=startp[tr]; /* this won't be 0 for tr. 0 & 1 : it's a pointer */
    if(size > 1024) {
        size+=a; /* make size look like a pointer */
        for(;a < (size-1024); a+=1024) {
            if(read(fd,vir_buff,1024) == -1)
                return(-1);
            _lmove(1024,vir_buff,_showds(),a,segment[tr]);
        }
    }
    else
        size+=a;

    if(!(size-a))
        return(1);
    if(read(fd,vir_buff,size-a) == -1)
        return(-1);
    _lmove(size-a,vir_buff,_showds(),a,segment[tr]);
    return(1);
}

seg_write(fd,tr,size)
int fd, tr; unsigned size;
{
    unsigned a;
    a=startp[tr]; /* this won't always be 0 because of tracks 0 & 1 */
    if(size > 1024) {
        size+=a; /* make size look like a pointer */
        for(;a < (size-1024); a+=1024) {

```



```

        _lmove(1024,a,segment[tr],vir_buff,_showds());
        if(write(fd,vir_buff,1024) == -1)
            return(-1);
    }
}
else
    size+=a;

if(!size-a) /* Is the buff length 0 || have we written all data? */
    return(1);
_lmove(size-a,a,segment[tr],vir_buff,_showds());
if(write(fd,vir_buff,size-a) == -1)
    return(-1);
return(1);
}

/* play() -
This function plays through the 16 tracks until track 0 ends
or a key is struck.
*/
play()
{
    int playing;
    char a,ch,b;
    unsigned n;
    scr_rowcol(2,0);
    printf("Tempo:   %d",r_tempo);

sp:
    for (a=0; a<15; ++a) {
        if(buffon[a] && a!=rbuff) {
            if(playfrom(a)) { /* Track or song end */
                if(a) /* if not track 0 */
                    buffon[a]=0; /* turn off buff */
                else
                    return;
            }
        }
        if(dsyc)
            bt();
        if(MIDIsyc)
            bt2();
        if(audmet)
            bt3();
    }
    if(!(ch=scr_csts()))
        goto sp;
    else {
        switch(ch) {
            case '+':
                ++r_tempo;
                break;
            case '-':
                --r_tempo;
                break;
            default:
                return;
        }
        scr_rowcol(2,10);
        printf("%d",r_tempo);
        tempo=2000000/((r_tempo*48)/60);
        set_cnt_1(tempo,tempo >> 8);
    }
    goto sp;
}

playfrom(a)
char a;
{
    char *p,d,*p1;
    p=vir_buff;
    _lmove(8,pointers[a],segment[a],vir_buff,_showds());

    if(*p==0xff) /* check for EOS */
        return(1);
}

```

(continued)

```

readclk();
p1=p+2;
if(*p1 > cmsb)
    goto pnow;
if(*p1 < cmsb)
    return(0);
if(*(p+1) < clsb)
    return(0);
pnow: /* now we know that it's time to play [notes]... */
d=*p & 0xc0;
switch(d)
{
    case 00: /* note on or off */
        pnt(p,parts[a].chan,parts[a].transpose);
        pointers[a]+=4;
        break;
    case 0x40: /* prog chan / chan pres. / key pres. */
        pchan(p,parts[a].chan);
        pointers[a]+=4;
        break;
    case 0x80: /* pitch bender */
        w_f_e(); /* let FIFO empty */
        _outb(0xE0+parts[a].chan,MIDID); /* send P. B. code */
        w_f_e(); /* let FIFO empty */
        _outb(0,MIDID); /* we didn't store LSB */
        w_f_e();
        _outb(*p << 1,MIDID); /* send MSB */
        pointers[a]+=3;
        break;
    case 0xC0: /* cntrl chan | temp chan */
        cchan(p,parts[a].chan);
        if(!(*p&1)); /* make up for control change */
            ++pointers[a];
        pointers[a]+=4;
        break;
}
return(0);
}

```

```

pnt(p,tc,trans)
char *p,tc,trans;
{

```

```

# asm

```

```

    MOV SI,WORD [BP+4] ; get p
    MOV AL,BYTE [SI+3] ; get *(p+3)
    AND AL,128 ; strip off top bit
    JZ znoff ; play a note off
    MOV AL,BYTE [BP+6] ; get tchan
    ADD AL,090H ; add a basic note on
zcheat1: ; here's where we'll cheat on a Note OFF
    MOV DX,0FFA0H ; MIDI Data port
    CALL w_f_e_ ; wait for DART FIFO to empty
    OUT DX,AL ; send byte
    MOV AL,BYTE [SI+3] ; get note to turn on
    ADD AL,BYTE [BP+8] ; add transpose val.
    AND AL,07FH ; strip off top bit
    CALL w_f_e_ ; wait for DART FIFO to empty
    OUT DX,AL ; send out note
    MOV AL,BYTE [SI] ; get velocity
    SHL AL,1 ; shift up into range
    CALL w_f_e_ ; wait for DART FIFO to empty out
    OUT DX,AL ; send velocity
    JMP zalldone ; finished

znoff: ; send Note OFF command
    MOV AL,BYTE [BP+6] ; get transmit channel
    ADD AL,080H ; add basic Note OFF
    JMP zcheat1 ; let's take a short cut

zalldone: ; all done

```

```

#
}

```

```

pchan(p,tc) /* Program change, channel pressure, or after-touch */
char *p,tc;
{

```



```

# asm
MOV DX,0FFA0H           ; DART port
MOV SI,WORD [BP+4]       ; get p
MOV AL,BYTE [SI]         ; get I.D. byte
CMP AL,040H             ; I.D. for Prog change
JZ zpchan               ; branch if Prog change
CMP AL,041H             ; I.D. for Chan pressure, after-touch
JZ zchanp               ; else, Poly. key press. (after-touch)

MOV AL,BYTE [BP+6]       ; get transmit chan
ADD AL,0A0H             ; poly. key pressure
CALL w_f_e_             ; send 1st byte
OUT DX,AL               ; get key #
MOV AL,BYTE [SI+3]
CALL w_f_e_             ; get pressure
OUT DX,AL               ; strip off bottom 5
MOV AL,BYTE [SI]         ; get pressure
AND AL,01FH             ; strip off bottom 5
SHL AL,1                ; bump up two bits
SHL AL,1
CALL w_f_e_             ; all done
OUT DX,AL
JMP zalldone2

zpchan:                 ; program change
MOV AL,BYTE [BP+6]       ; get transmit channel
ADD AL,0C0H             ; basic prog chan
JMP zcheat2             ; use code from above routine

zchanp:                 ; channel pressure
MOV AL,BYTE [BP+6]       ; get transmit channel
ADD AL,0D0H             ; add basic chan pres
JMP zcheat2

zcheat2:                ; entry point from zpchan
CALL w_f_e_             ; wait for DART
OUT DX,AL               ; send byte
MOV AL,BYTE [SI+3]       ; get new vel
OUT DX,AL               ; send

zalldone2:              ; FINE
#
#
cchan(p,tc)
char *p,tc;
# asm
MOV SI,WORD [BP+4]       ; get p
MOV AL,BYTE [SI]         ; get I.D. byte
AND AL,1                ; bit 0
JNZ ztchan              ; tempo change
                           ; now we know it's a cntrl chan
MOV AL,0B0H             ; cntrl chan
ADD AL,BYTE [BP+6]       ; add transmit channel
MOV DX,0FFA0H           ; MIDID port
CALL w_f_e_             ; wait for DART to empty
OUT DX,AL
MOV AL,BYTE [SI+3]       ; get cntrl #
CALL w_f_e_             ; wait for DART FIFO to empty
OUT DX,AL
MOV AL,BYTE [SI+4]       ; get cntrl val
CALL w_f_e_
OUT DX,AL
JMP _fine2              ; done

ztchan:                 ; tempo change
; MOV AL,extsync_
; OR AL,AL
; JNZ _fine2            ; set flags
                           ; if extsync mode, then this has no purpose
#
/* tempo(*(p+3)+30); */
# asm

```

(continued)

June

_fine2:

```
#  
}
```

w_f_e() /* this function "sleeps" until the FIFO in the DART is empty */

```
# asm
```

```
PUSH DX  
PUSH AX  
MOV DX,0FFA2H          ; DART status register
```

try_again:

```
IN AL,DX  
AND AL,4               ; Tx buff empty?  
JZ try_again          ; no?  
POP AX  
POP DX
```

```
#  
}
```

```
/*
```

```
BT, BT2, BT3  
These are the meternome counting routines.
```

```
*/
```

bt()

```
{
```

```
readclk();  
if(beat==clsb) {  
    _outb(5,MIDIS+1); /* send sync high */  
    _outb(128,MIDIS+1);  
    _outb(5,MIDIS+1); /* send sync low */  
    _outb(0,MIDIS+1);  
    beat-=s_t_b;  
}
```

```
}
```

bt2()

```
{
```

```
readclk();  
if(mbeat==clsb) {  
    _outb(0xf8,MIDID); /* MIDI timing clk */  
    mbeat-=m_t_b;  
}
```

```
}
```

bt3()

```
{
```

```
readclk();  
if(abeat==clsb) {  
    if(printer)  
        beep();  
    abeat-=a_t_b;  
}
```

```
}
```

beep()

```
{
```

```
_rax=0x0500;  
_rdx=7;  
_doint(0x21);
```

```
}
```

```
/*
```

```
SETINT  
This routine sets up the interrupt.
```

```
*/
```

setint()

```
{
```

```
int rxint(),a;  
int seg_num, (*fun_add)();  
  
fun_add = rxint;  
seg_num=_showcs();  
_poke(fun_add & 0xff,0x28,0); _poke(fun_add >> 8,0x29,0);  
_poke(seg_num & 0xff,0x2a,0); _poke(seg_num >> 8,0x2b,0);  
setdart();  
setpic();
```

```
}
```



```

setdart()
{
    _outb(24,MIDIS); /* reset channel A */
    _outb(1,MIDIS);  _outb(24,MIDIS); /* int on all Rx + Ext Stat */
    _outb(3,MIDIS);  _outb(193,MIDIS);
    _outb(4,MIDIS);  _outb(196,MIDIS);
    _outb(5,MIDIS);  _outb(104,MIDIS);
}

setpic()
{
    char a;
    a=_inb(PIC1); /* read interrupt mask */
    if(a&4) /* if DART is masked off */
        a-=4;
    _outb(a,PIC1); /* re-enable ints */
}

setcounter(l1,m1,l2,m2)
char l1,m1,l2,m2;
{
    char e,d;
    _outb(0x34,CSTAT); /* counter 1=mode 2 - read/load both */
    e=d; d=e;
    _outb(0x70,CSTAT); /* counter 2= mode 0 - read/load both */
    e=d; d=e;
    _outb(l2,COUNTER2); /* set LSB of counter 2 */
    e=d; d=e;
    _outb(m2,COUNTER2); /* set MSB of counter 2 */
    e=d; d=e;
    _outb(l1,COUNTER1); /* set LSB of counter 1 */
    e=d; d=e;
    _outb(m1,COUNTER1); /* set MSB of counter 1 */
    readclk();
    while(cmsb!=255 && clsb!=255)
        readclk();
    return;
}

set_cnt_1(l1,m1)
char l1,m1;
{
    _outb(l1,COUNTER1); /* set LSB of counter 1 */
    _outb(m1,COUNTER1); /* set MSB of counter 1 */
}

intoff()
{
    _outb(01,MIDIS); /* turn off my interrupt */
    _outb(00,MIDIS); /* (at the DART) */
}

inton()
{
    _outb(01,MIDIS); /* turn on my interrupt */
    _outb(24,MIDIS); /* (at the DART) */
}

readclk()
{
    # asm
        MOV AL,64
        MOV DX,0FFA7H
        CLI ; an interrupt right now from Rxint would be bad
        OUT DX,AL
        NOP
        NOP
        NOP
        SUB DX,2
        IN AL,DX
        MOV clsb_,AL
        NOP
        NOP
        NOP
        IN AL,DX
        MOV cmsb_,AL
        STI ; ints back on
    #
}

```

(continued)

```

scr_cla() /* clear screen & psn cursor at top */
{
    scr_clr();
    scr_rowcol(0,0);
}

/* MMODE
   This is a no-frills routine for setting up various
   MIDI system modes
*/
mmode()
{
    char a,mo,v;
st:
    scr_cla();
    printf("                                MIDI mode assignments:\n");
    printf("%s\n",sin_line);
ss:
    putchar('\n');
    printf("      1 - Omni ON   / Poly\n");
    printf("      2 - Omni OFF  / Poly\n");
    printf("      3 - Omni ON   / Mono\n");
    printf("      4 - Omni OFF  / Mono\n");
    printf("      5 - Program change\n");
    printf("      6 - All notes off (all channels)\n");
    printf("      7 - All notes off (one channel)\n");
    printf("      0 - Exit\n");
    printf("Enter new mode:");
    mo=getint();
    if(!mo)
        return;
    if(mo>7)
        goto st;
    if(mo==6)
        goto skip;
    printf("Send over which channel:");
    a=getint();
    if(a > 15)
        goto ss;
    printf("\nChannel %d:\n",a);
skip:
    switch(mo)
    {
        case 1:
            printf(" Poly / Omni on\n");
            poly(a);
            o_on(a);
            break;
        case 2:
            printf(" Poly / Omni off\n");
            poly(a);
            omni_off(a);
            break;
        case 3:
            printf(" Mono / Omni on\n");
            mono(a);
            o_on(a);
            break;
        case 4:
            printf(" Mono / Omni off\n");
            mono(a);
            omni_off(a);
            break;
        case 5:
            printf(" Program change\n");
            printf(" Enter new program: ");
            v=getint();
            _outb(192+a,MIDIID);
            _outb(v,MIDIID);
            break;
        case 6:
            for(a=0; a<16; ++a)
            {
                _outb(0xb0+a,MIDIID);
                _outb(123,MIDIID);
                w_f_e();
                _outb(0,MIDIID);
            }
            printf(" All notes off - all channels\n");
    }
}

```



```

        getchar();
        break;
    case 7:
        _outb(0xb0+a,MIDID);
        _outb(123,MIDID);
        w_f_e();
        _outb(0,MIDID);
        printf(" All notes off\n");
        getchar();
        break;
    }

    goto st;
}

o_on(ch)
char ch;
{
    _outb(176+ch,MIDID);
    _outb(125,MIDID);
    w_f_e();
    _outb(0,MIDID);
    getchar();
}

omni_off(ch)
char ch;
{
    _outb(176+ch,MIDID);
    _outb(124,MIDID);
    w_f_e();
    _outb(0,MIDID);
    getchar();
}

poly(ch)
char ch;
{
    _outb(176+ch,MIDID);
    _outb(127,MIDID);
    w_f_e();
    _outb(0,MIDID);
    getchar();
}

mono(ch)
char ch;
{
    char v;
    printf("How many channels? (0 = # in receiver, or other) ");
    v=getint();
    _outb(176+ch,MIDID);
    _outb(126,MIDID);
    w_f_e();
    _outb(v,MIDID);
    getchar();
}

get_av_mem() /* returns total system RAM - in K */
{
    _doint(0x012);
    return(_rax);
}

char *_showss()
{
    # asm
    MOV AX,SS
    #
}

hak() /* display "hit any key to continue" & wait */
{
    puts("\n\n");
    scr_cls();
    puts("
Hit any key to continue.");
}

```

(continued)

```

        while(!scr_csts());
    }

getint()      /* error check an input line & return int */
{
    char inln[20],*b,flag;
stg:
    _gets(inln,20);
    if(strlen(inln))
        for(b=inln;*b!='\0';++b)
            if(!isdigit(*b) && *b != '-' )    {
                putchar(7);
                goto stg;
            }
        return(atoi(inln));
    }
    else
        putchar(7);
    goto stg;
}

/*      ED_IN_FILT
This routine allows the user to edit the input filter
*/
ed_in_filt()
{
    int n;
    scr_clea();
    scr_rowcol(0,26);
    printf("Edit Input Filter\n");
    printf("%s\n\n",sin_line);
st_filt:
    scr_rowcol(3,0);
    printf("
    Mess #:      Message type:      Status:\n");
    printf("
    1      T      Note ON           T      %s\n",(in_filt & 1) ? yes : no);
    printf("
    2      T      Note OFF          T      %s\n",(in_filt & 2) ? yes : no);
    printf("
    3      T      Program Change    T      %s\n",(in_filt & 4) ? yes : no);
    printf("
    4      T      Channel Pressure  T      %s\n",(in_filt & 8) ? yes : no);
    printf("
    5      T      Pitch Wheel       T      %s\n",(in_filt & 16) ? yes : no);
    printf("
    6      T      Control Change    T      %s\n",(in_filt & 32) ? yes : no);
    printf("
    7      T      Poly. Key Pressure T      %s\n",(in_filt & 64) ? yes : no);
    printf("
    aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\n");
    printf("\n\n\n\n\n");
re_try:
    scr_rowcol(18,12);
    printf("Enter Message Number to toggle, 0 to end: ");
    n=getint();
    if(n<0 || n>7)
        goto re_try;
    if(!n)
        return;
    n=1 << (n-1); /* make n into bit to complement */
    in_filt = in_filt & n ? in_filt-n : in_filt+n;
    goto st_filt;
}

get_options() /* enter various user options */
{
em:
    printf("Enter metrenome speed:");
    r_tempo=getint();
    if(r_tempo<40 || r_tempo>200)
        goto em;
    tempo=200000/((r_tempo*48)/60);
    printf("Audio Sync (Y/N):");
    audmet=(toupper(getchar())=='Y') ? 1:0;
    putchar('\n');
    printf("Drum Sync (Y/N):");
    dsync=(toupper(getchar())=='Y') ? 1:0;
    putchar('\n');
    printf("MIDI Sync (Y/N):");
    MIDIsync=(toupper(getchar())=='Y') ? 1:0;
}

```



```

/* dir(fname,ext)
   char *fname,*ext;

   This function shows a directory of the currently logged directory
   and disk and print it on the screen in "five-across" format.
*/

dir(fname,ext) /* function to show directory of currently logged disk & dir */
char *fname,*ext; /* pointers to filename & extension */
{
    struct fcb {
        char drive_num;
        char filename[8];
        char extension[3];
        char rest[25]; /* the rest of the fcb */
    } fcb;

    struct fcb fcb2;
    unsigned dta_add, dta_seg;
    char filename[21];
    char flag=4;
    char cnt=80;

    fcb.drive_num=0;
    strcpy(filename,"");
    strncpy(fcb.filename,fname,8);
    strncpy(fcb.extension,ext,3);
    _setmem(fcb.rest,25,0);

    _rax=0x2f00; /* get current DTA */
    _doint(0x21);
    dta_add=_rbx;
    dta_seg=_res; /* this is the offset & segment of the current DTA */
    _rdx=fcb;
    _rds=_showds();
    _rax=0x1100;

    scr_rowcol(6,0);
    printf("%s\n",sin_line);
    scr_rowcol(7,0);
    scr_cls();
    for(;;) {
        _doint(0x21);
        if((_rax & 0xff))
            break; /* last file has been listed */
        _lmove(20,dta_add,dta_seg,fcb2,_showds());
        strncpy(filename,fcb2.filename,8);
        filename[8]='.';
        filename[9]='\0';
        strcat(filename,fcb2.extension);
        filename[12]='\0';
        puts(filename);
        puts(" ");
        if(!flag)
            flag=5;
        if(!--cnt) {
            printf("Hit any key to continue.");
            getchar();
            scr_rowcol(7,0);
            scr_cls();
            cnt=80;
            flag=4;
        }
        --flag;
        _rax=0x1200;
    }
}

```

rxint11.a

TEXT

"A MIDI Project," by Jay Kubicky.
June, page 199. Also download midi111.c.

```

0      RXINT:
;      version 1.1
;      2/2/86
;
;      This is the main receive interrupt.
;      It is called when the DART receives data.
;      This version supports carry-over (running) status bytes
;      as well as a translate table.
;

DSEG

; These are C's globals
PUBLIC r_segment_
PUBLIC ptr_
PUBLIC end_
PUBLIC destbyte_
PUBLIC clsb_
PUBLIC cmsb_
PUBLIC stop_
PUBLIC LAST_STAT_
PUBLIC C_O_F_
PUBLIC clk_type_
PUBLIC counter_dec_
PUBLIC in_filt_

; status carry-over flag
; 0=internal, 1=external

; the MIDI input filter
; bit:
; 0 = Note on filt
; 1 = Note off filt
; 2 = Prog change filt
; 3 = Channel after-touch
; 4 = Pitch wheel
; 5 = All other controllers
; 6 = Key after-touch

CSEG      ; ALL CODE

PUBLIC rxint_

rxint_:
STI      ; interrupts back on
PUSH AX
MOV AL,0B9H
OUT 021H,AL      ; shut off counter interrupt
PUSH BX
PUSH DX
PUSH ES
PUSH DS
MOV AX,0
; this will be new value for the
; DS so we can access the OLD DS

MOV DS,AX
MOV BX,WORD [04FAH]      ; put new DS in BX
MOV DS,BX      ; put new DS in DS
MOV AX,r_segment_
MOV ES,AX      ; set buffer segment

sst:
MOV DX,0FFA2H
MOV AL,1
OUT DX,AL
IN AL,DX
AND AL,32
JNZ finished
; midi stat
; rd register 1
; RD reg 1
; Rx overrun error?
; if so, than we're REALLY done!
; jumped to by _iret if another char is

_st:
available
IN AL,DX
AND AL,1
JZ _iret
; read status register
; Rx char?
; if not, return

```



```

MOV DX,0FFA0H      ; MIDI data
IN AL,DX           ; now let's read it
MOV DL,AL          ; store in DL for later
CMP AL,0F8H        ; timing clock?
JNZ C1             ; if not, check for stop
MOV DL,counter_dec_ ; get amount to decrement counter
MOV DH,0           ; we'll do a word subtract
MOV AL,clsb_       ; get lsb
MOV AH,cmsb_       ; get msb
SUB AX,DX          ; dcr word
MOV clsb_,AL       ; store lsb
MOV cmsb_,AH       ; store msb
JMP _iret          ; if is, all done

C1:
CMP AL,0FCH        ; MIDI Stop?
JNZ C4             ; if not, continue
MOV BYTE stop_,1   ; else, set stop byte
JMP _iret          ; and return

C4:
AND AL,0F0H        ; strip off top 4
CMP AL,0F0H        ; running status?
JZ _iret           ; if so, we're done
MOV BYTE AL,destbyte_ ; get destbyte
CMP AL,0FFH        ; shall we continue?
JZ _iret           ; if not, then we're done
MOV BL,AL          ; keep for later
AND AL,0F0H        ; strip off top four
JZ _newmsg         ; first byte of a new message - What a
revelation!
contzz:            ; branch point from carry-over status
CMP AL,010H        ; Note ON?
JZ _nton           ; yes?
CMP AL,020H        ; Note OFF?
JZ _noff           ; yes?
CMP AL,070H        ; Poly key pres?
JZ _keypres        ; Bender?
CMP AL,040H        ; yes?
JZ _bender         ; Control Change?
CMP AL,050H        ; yes?
JZ _cchan

CMP AL,060H        ; Channel Velocity?
JZ _cvel           ; For now, throw away channel velocity
CMP AL,030H        ; Prog Change?
JZ _pchan          ; yes?
JMP _iret          ; This would fall through for FILT'd out codes

; this routine (newmsg) processes the first byte of a given
; message, and also decides if it is an important message

_newmsg:            ; it's a new message
MOV AL,DL           ; retrieve
AND AL,080H
MOV AL,DL
MOV BYTE C_O_F_,0   ; no carry-over
JNZ contxx          ; check for MIDI carry-over stat.
MOV BYTE AL,LAST_STAT_ ; get previous status byte
MOV BYTE C_O_F_,0FFH ; carry-over status

contxx:
MOV BYTE destbyte_,0 ; this is default in case code isn't
                     ; supported or is filtered out
MOV CL,in_filt_     ; this is input filter
AND AL,0F0H         ; strip off channel info.
MOV LAST_STAT_,AL   ; save new last stat
CMP AL,090H         ; Note ON?
JZ _nnon            ; yes?
CMP AL,080H         ; Note OFF?
JZ _nnoff           ; yes?
CMP AL,0A0H         ; Poly key press?
JZ _nkeypres        ; yes?
CMP AL,0E0H         ; Pitch Wheel
JZ _npwch           ; yes?
CMP AL,0B0H         ; Control Change?
JZ _ncchan          ; yes?
CMP AL,0C0H         ; Prog. Chan?

```

(continued)

```

JZ _npchan                ; yes?
CMP AL,0D0H               ; Channel Velocity?
JZ _nchanv               ; yes?
JMP _iret                ; otherwise, it must be a code we don't
support

_nnon:                    ; first byte of a note on just came in
AND CL,1                  ; keep note-ons?
JZ _iret                  ; no, then all done.
CALL _stime               ; store present time
MOV BYTE ES:[BX],0        ; put 0 at *ptr - I.D. code for note on/off
MOV BYTE destbyte_,011H   ; note on in dest byte
JMP _cco                  ; check carry-over

_nnoff:                   ; first byte of note off just came in
AND CL,2                  ; keep note-offs?
JZ _iret                  ; no, then all done
CALL _stime               ; same as _nnon (above)
MOV BYTE ES:[BX],0        ; I.D.
MOV BYTE destbyte_,021H   ; note off in destbyte
JMP _cco                  ; check carry-over

_nchan:                   ; first byte of control change
AND CL,32                 ; keep control changes?
JZ _iret                  ; no
CALL _stime               ; I.D.
MOV BYTE ES:[BX],0C0H     ; cchan in destbyte
MOV BYTE destbyte_,051H   ; check carry-over
JMP _cco

_npchan:                  ; 1st byte of program change
AND CL,4                  ; keep prog. changes?
JZ _iret
CALL _stime               ; I.D.
MOV BYTE ES:[BX],040H     ; prog chan in destbyte
MOV BYTE destbyte_,031H   ; check carry-over
JMP _cco

_nchanv:                  ; 1st byte of Channel pressure
AND CL,8                  ; keep channel pressure?
JZ _iret
CALL _stime               ; I.D.
MOV BYTE ES:[BX],041H     ; chan vel in dbyte
MOV BYTE destbyte_,061H   ; check carry-over
JMP _cco

_nkeypres:                ; 1st byte of Poly. key pres.
AND CL,64                 ; keep poly. key pres?
JZ _iret
CALL _stime               ; I.D.
MOV BYTE ES:[BX],060H     ; chan vel in dbyte
MOV BYTE destbyte_,071H   ; check carry-over
JMP _cco

_npwhch:                  ; first byte of pitch wheel change
AND CL,16                 ; keep pitch wheel?
JZ _iret
CALL _stime               ; I.D.
MOV BYTE ES:[BX],080H     ; pitch wheel in dbyte
MOV BYTE destbyte_,041H   ; check carry-over
JMP _cco

_cco:                     ; check carry over flag
MOV BYTE AL,C_O_F_        ; get carry over flag
OR AL,AL                  ; set flags
JZ _iret
MOV BYTE AL,destbyte_
MOV BL,AL
AND AL,0F0H
JMP contzz                ; carry over

; now for the routines that are called while a message is in
; progress:

```



```

_nton:
    AND BL,1
    MOV WORD BX,ptr_
    JZ _ntonvel
    OR DL,080H
    ADD BX,3
    MOV BYTE ES:[BX],DL
    INC BYTE destbyte_
    JMP _iret
; note on processing routine
; BL still has destbyte in it
; get pointer
; must be a 2, so go and save velocity
; set top bit of note
; inr pointer to note
; store it
; inr for velocity, which will come in next
; all done

_ntonvel:
    SHR DL,1
    OR BYTE ES:[BX],DL
    MOV BYTE destbyte_,0
    ADD WORD ptr_,4
    JMP _iret
; must be a velocity byte
; shift velocity byte one to the right
; OR it in
; next byte will be a newmsg
; inr ptr
; all done

_noff:
    AND BL,1
    MOV WORD BX,ptr_
    JZ _ntoffvel
    ADD BX,3
    MOV BYTE ES:[BX],DL
    INC BYTE destbyte_
    JMP _iret
; note off processing routine
; BL still has destbyte in it
; get pointer into buff
; if destbyte=2, then it must be a velocity
; inr to where we'll store the note
; store note; top bit should already be 0
; inr destbyte for velocity
; all done

_ntoffvel:
    SHR DL,1
    OR BYTE ES:[BX],DL
    MOV BYTE destbyte_,0
    ADD WORD ptr_,4
    JMP _iret
; must be a velocity byte
; shift DL logically right 1
; OR it in
; next byte will be a newmsg
; now _ptr points to the next message spot
; all done

_pchan:
    MOV WORD BX,ptr_
    ADD BX,3
    MOV BYTE ES:[BX],DL
    MOV BYTE destbyte_,0
    ADD WORD ptr_,4
    JMP _iret
; program change (new program)
; get pointer; no need to check byte number
; because only two bytes are transmitted
; this is where we'll store the new prog. #
; store data
; reset to newmsg
; inr to psn. of next message
; all done

_bender:
    AND BL,1
    JZ _bmsb
    INC BYTE destbyte_
    JMP _iret
; store bender data
; first data byte?
; if not, than go and store the MSB
; inr for next pass
; all done

_bmsb:
    SHR DL,1
    MOV WORD BX,ptr_
    OR BYTE ES:[BX],DL
    ADD WORD ptr_,3
    MOV BYTE destbyte_,0
    JMP _iret
; store MSB of bender
; shift DL down 1
; get ptr
; OR in top 6 of MSB of bender
; inr pointer to next message
; next message will newmsg
; all done

_cchan:
    AND BL,1
    MOV WORD BX,ptr_
    JNZ _cnum
    ADD BX,4
    MOV BYTE ES:[BX],DL
    MOV BYTE destbyte_,0
    ADD WORD ptr_,5
    JMP _iret
; control change
; control #?
; get ptr_
; if input is control #, then save it
; this is where we'll store the control value
; store control value
; next byte will newmsg
; inr to next message location
; all done

_cnum:
    ADD BX,3
    MOV BYTE ES:[BX],DL
    INC BYTE destbyte_
    JMP _iret
; number
; address of control number
; store control number
; next byte will be control value
; all done

_cvel:
    MOV WORD BX,ptr_
    MOV BYTE ES:[BX+3],DL
    ADD WORD ptr_,4
    MOV BYTE destbyte_,0
    JMP _iret
; channel velocity
; load BX w/pointer
; store velocity
; inr to next message psn.
; next byte will be new msg
; all done

```

(continued)

```

_keypres:                                ; poly. key pressure
      AND BL,1                            ; amount?
      MOV WORD BX,ptr_
      JZ _storepres                       ; store key pressure
      ADD BX,3                            ; this is where to store the note num.
      MOV BYTE ES:[BX],DL                ; store it
      INC BYTE destbyte_                 ; next byte will be val.
      JMP _iret

_storepres:                              ; store key pressure
      SHR DL,1
      SHR DL,1                            ; shift DL right 2 (/4)
      OR ES:[BX],DL                      ; OR it in
      MOV BYTE destbyte_,0               ; next byte new message
      JMP _iret                          ; all done

_stime:                                  ; routine to read the PIT and store the
results
      PUSH DX                            ; in the buffer
      MOV WORD BX,ptr_                   ; preserve data
      MOV AL,clk_type_                   ; get pointer
      OR AL,AL                           ; get clk type (0=int)
      JNZ ext_clk                        ; set flags
      MOV AL,64                          ; is clk_type=ext_clk??
      MOV DX,0FFA7H                      ; counter latching operation
      OUT DX,AL                          ; counter stat
      NOP                                ; out to PIT
      NOP                                ; stall for time
      MOV DX,0FFA5H                      ; counter 2
      IN AL,DX                           ; read _clsb
      MOV BYTE ES:[BX+1],AL              ; store in buffer

      NOP
      NOP
      IN AL,DX                           ; read _cmsb
      MOV BYTE ES:[BX+2],AL
      POP DX                             ; restore data
      RET

ext_clk:                                ; timing clock from MIDI
      MOV DL,clsb_
      MOV DH,cmsb_                       ; get MSB & LSB
      MOV BYTE ES:[BX+1],DL
      MOV BYTE ES:[BX+2],DH              ; store MSB & LSB
      POP DX                             ; restore DX
      RET                                ; all done

_eoi:                                    ; send EOI to PIC
      PUSH AX                            ; we'll be using this
      MOV AL,020H
      OUT 020H,AL                        ; EOI
      POP AX                             ; EOI
      RET                                ; restore AX
                                           ; all done

_iret1:                                  ; execute EOI and then return
      CALL _eoi

finished:                                ; Rx overrun error
      MOV BYTE stop_,0FFH                ; CRITICAL ERROR!!!
      JMP f2

_iret:                                   ; all done routine
      MOV DX,0FFA2H                      ; DART status reg
      IN AL,DX
      AND AL,1                           ; RxD? (THIS WILL HAVE TO BE CHANGED TO CHECK
                                           ; FOR EXTERNAL STAT. ALSO)
      JNZ sst                            ; if so, than go to top of routine
f2:                                       ; C equivalent:
                                           ; if(end > ptr)
                                           ; goto okay;
                                           ; else
                                           ; stop=1;

      MOV BX,ptr_
      MOV DX,end_
      CMP DX,BX

```



```

; JG f3
; MOV BYTE stop_,1 ; Waaaaah!! Stop Everything!! - We're out of memory
f3:
    mov al,0b8h
    out 021h,al
    POP DS
    POP ES
    POP DX
    POP BX
    POP AX ; retrieve registers
    call _eoi

    IRET ; ALL DONE!!!!!!

```

macview.pas

TEXT
 "Decoding MacPaint on the IBM PC" by Mark Anacker.
 June, page 131

```

(*)
    MACVIEW.PAS - Display MacPaint pictures on the IBM PC
                  graphics adapter.

                  Mark Anacker    09/24/85
                                     *)

PROGRAM MACVIEW;
(*$V-*)

TYPE
    STRING255 = STRING[255];
    SCANLINE = ARRAY [1..80] OF BYTE;    (* hi-res screen is 80 bytes wide *)

CONST
    SCANFLAG : BOOLEAN = TRUE;
    ODDLINE  : INTEGER = 0;
    EVENLINE  : INTEGER = 0;

VAR
    FILNAM : STRING[64];
    PNTFIL : FILE OF BYTE;
    ELEN   : BYTE;
    ELEM   : STRING255;
    OUTPAT : STRING255;
    LCNT   : INTEGER;
    CNT    : INTEGER;
    XPOS,YPOS : INTEGER;
    STLINE,ENLINE : INTEGER;
    EVSCREEN : ARRAY [0..99,1..80] OF BYTE ABSOLUTE $B800:$0000;
    ODScreen : ARRAY [0..99,1..80] OF BYTE ABSOLUTE $B800:$2000;

    DOIT : BOOLEAN;

    SLPTR : ARRAY [0..719] OF ^SCANLINE;    (* pointers to scan line buffers *)

PROCEDURE HELPMMSG;
BEGIN
    WRITE('MACVIEW - View MacPaint images ... by Mark Anacker');
    WRITELN(' 09/24/85'); WRITELN;
    WRITELN('This program will let you view an entire MacPaint image.'):
    WRITELN('First, transfer the picture file from the Mac. Then,'):
    WRITELN('run this program and give it the file name. You may use'):
    WRITELN('the keypad keys 1-3 and 7-9 to scroll over the image.'):
    WRITELN('Press the space bar to exit back to DOS.'):
    WRITELN;
    WRITELN('You may specify the file name on the command line.'):
    WRITELN;
    WRITELN('Remember, the aspect ratio is different. The picture will'):
    WRITELN('be distorted somewhat in the vertical direction.'):
    WRITELN;
END;

```

(continued)

```

PROCEDURE GETFILE;
VAR   CNT : INTEGER;
      INCH : CHAR;
BEGIN
  ASSIGN(PNTFIL,FILNAM);
  (*$I-*) RESET(PNTFIL); (*$I+*)
  IF IORESULT<>0 THEN
    BEGIN
      WRITELN('** Error opening file - halting **'); HALT(1);
    END;
  SEEK(PNTFIL,512);          (* skip brush patterns *)
  YPOS:=0; LCNT:=0; OUTPUT:='';
  SCANFLAG:=TRUE;
  ODDLINE:=0; EVENLINE:=0;
  STLINE:=0;
END;

PROCEDURE OUTSCREEN;          (* display line *)
VAR   CNT,CNT2 : INTEGER;
BEGIN
  FOR CNT:=1 TO LENGTH(OUTPUT) DO
    OUTPUT[CNT]:=CHR(NOT ORD(OUTPUT[CNT])); (* invert bits to black *)
    NEW(SLPTR[LCNT]); (* on white like the Mac *)
    (* allocate a new buffer line*)
    FILLCHAR(SLPTR[LCNT]^,80,CHR(0)); (* fill it to black *)
    MOVE(OUTPUT[1],SLPTR[LCNT]^,72); (* and copy the decoded bits *)
  END;

PROCEDURE PUTLINE;           (* decide if we need to do line *)
BEGIN
  OUTPUT:=OUTPUT+ELEM;      (* build scan line pattern *)
  IF LENGTH(OUTPUT)>=72 THEN (* if we have a full line, *)
    BEGIN
      IF LENGTH(OUTPUT)>72 THEN (* if too long, truncate *)
        OUTPUT:=COPY(OUTPUT,1,72);
      OUTSCREEN; (* put it in the buffer *)
      LCNT:=LCNT+1; OUTPUT:='';
      FILLCHAR(OUTPUT,75,CHR(0)); (* reset the string *)
    END;
  END;

PROCEDURE REPBLOCK;          (* block of repeating data *)
VAR   TMPBYTE : BYTE;
      CNT : INTEGER;
BEGIN
  ELEN:=(256-ELEN); (* get character count *)
  READ(PNTFIL,TMPBYTE); (* get character to repeat *)
  ELEM:='';
  FOR CNT:=0 TO ELEN DO
    ELEM:=CONCAT(ELEM,CHR(TMPBYTE)); (* make string of chars. *)
  PUTLINE; (* test for a complete scan line *)
END;

PROCEDURE MIXBLOCK;          (* block of mixed, raw data *)
VAR   TMPBYTE : BYTE;
      CNT : INTEGER;
BEGIN
  ELEM:='';
  FOR CNT:=0 TO ELEN DO
    BEGIN
      READ(PNTFIL,TMPBYTE); (* get characters *)
      ELEM:=CONCAT(ELEM,CHR(TMPBYTE)); (* add to running pattern *)
    END;
  PUTLINE; (* test for complete scan line *)
END;

PROCEDURE LOADBUF;           (* read data from file *)
BEGIN
  GETFILE; (* open file *)
  WRITELN('Loading picture into buffer ... Please wait a moment');
  REPEAT
    BEGIN
      READ(PNTFIL,ELEN); (* get a byte *)
      IF ELEN>127 THEN (* if 8th bit set, *)
        REPBLOCK (* it's a repeater, else *)
      ELSE
        MIXBLOCK; (* it's mixed *)
    END;
  UNTIL IORESULT=0;
END;

```



```

    END;
    UNTIL LCNT>=720;          (* until all 720 lines are done *)
    CLOSE(PNTFIL);
END;

PROCEDURE SHOWBUF;          (* display the buffer on screen *)
VAR CNT : INTEGER;
BEGIN
    EVENLINE:=0; ODDLINE:=0;
    FOR CNT:=STLINE TO STLINE+199 DO (* show the current 200 scan lines *)
        BEGIN
            IF (CNT AND 1)<>1 THEN      (* even line *)
                BEGIN
                    MOVE(SLPTR[CNT]^,EVSCREEN[EVENLINE,1],80); (* even lines *)
                    EVENLINE:=EVENLINE+1;
                END
            ELSE
                BEGIN (* odd line *)
                    MOVE(SLPTR[CNT]^,ODSCREEN[ODDLINE,1],80); (* odd lines *)
                    ODDLINE:=ODDLINE+1;
                END;
            END;
        END;
    END;

PROCEDURE MOVEIT;          (* scroll the picture up and down *)
CONST
    FKTABLE : STRING[6] = 'OPOGHI'; (* cursor/numeric key *)
    FKEQUIV : STRING[6] = '123789'; (* conversion table *)
VAR INCH : CHAR;
    CNT : INTEGER;
BEGIN
    REPEAT
        GOTOXY(75,1); WRITE(STLINE:3); (* put top scan line in upper corner *)
        READ(KBD,INCH); (* get the key from the user *)
        IF (INCH=CHR(27)) AND KEYPRESSED THEN (* if a real cursor key, *)
            BEGIN
                READ(KBD,INCH); (* get the key code *)
                IF POS(INCH,FKTABLE)>0 THEN (* and convert to it's number *)
                    BEGIN
                        CNT:=POS(INCH,FKTABLE); INCH:=FKEQUIV[CNT];
                    END;
                END;
            END;
        CASE INCH OF
            '8' : BEGIN (* move image UP *)
                IF STLINE<520 THEN
                    BEGIN
                        FOR CNT:=0 TO 98 DO
                            BEGIN
                                MOVE(EVSCREEN[CNT+1,1],EVSCREEN[CNT,1],72);
                                MOVE(ODSCREEN[CNT+1,1],ODSCREEN[CNT,1],72);
                            END;
                        FILLCHAR(EVSCREEN[99,1],72,CHR(0));
                        FILLCHAR(ODSCREEN[99,1],72,CHR(0));
                        STLINE:=STLINE+2;
                        MOVE(SLPTR[STLINE+198]^,EVSCREEN[99,1],72);
                        MOVE(SLPTR[STLINE+199]^,ODSCREEN[99,1],72);
                    END;
                END;
            '2' : BEGIN (* move image DOWN *)
                IF STLINE>1 THEN
                    BEGIN
                        FOR CNT:=99 DOWNT0 1 DO
                            BEGIN
                                MOVE(EVSCREEN[CNT-1,1],EVSCREEN[CNT,1],72);
                                MOVE(ODSCREEN[CNT-1,1],ODSCREEN[CNT,1],72);
                            END;
                        FILLCHAR(EVSCREEN[0,1],72,CHR(0));
                        FILLCHAR(ODSCREEN[0,1],72,CHR(0));
                        STLINE:=STLINE-2;
                        MOVE(SLPTR[STLINE]^,EVSCREEN[0,1],72);
                        MOVE(SLPTR[STLINE+1]^,ODSCREEN[0,1],72);
                    END;
                END;
            '3' : BEGIN (* page image DOWN *)
                STLINE:=STLINE-100;
            END;
        END;
    UNTIL FALSE;
END;

```

(continued)

```

        IF STLINE<0 THEN STLINE:=0;
        SHOWBUF;
    END;
    '9' : BEGIN                                (* page image UP *)
        STLINE:=STLINE+100;
        IF STLINE>520 THEN STLINE:=520;
        SHOWBUF;
    END;
    '7' : BEGIN                                (* go to TOP of image *)
        STLINE:=0; SHOWBUF;
    END;
    '1' : BEGIN                                (* go to BOTTOM of image *)
        STLINE:=514; SHOWBUF;
    END
END;
UNTIL INCH=' ';                                (* exit when SPACE bar is pressed *)
END;                                           (* move it *)

        (* main section *)

BEGIN
    TEXTCOLOR(7);                                (* set color to gray *)
    IF PARAMCOUNT=0 THEN                        (* If a blank command line, *)
        BEGIN
            HELPMMSG;                            (* show message, prompt for file *)
            WRITE('MacPaint file name : '); READLN(FILNAM);
            IF FILNAM='' THEN HALT(2);            (* if none given, exit to DOS *)
        END
    ELSE
        FILNAM:=PARAMSTR(1);                    (* else get file name from cmd line *)
        IF POS('.',FILNAM)=0 THEN                (* if no extension was given, *)
            FILNAM:=FILNAM+'.MCP';                (* assume .MCP *)
        LOADBUF;                                (* load the file *)
        HIRES;                                  (* switch to hi-res mode *)
        SHOWBUF;                                (* display the buffer *)
        MOVEIT;                                 (* move it up and down *)
        CLRSCR;                                 (* clear the screen when done *)
    END.

```


D·I·S·K·S A·N·D D·O·W·N·L·O·A·D·S

ORDERING DISKS OF BYTE LISTINGS

Listings that accompany BYTE articles are available in a variety of disk formats and on Cauzin Softstrip. Each disk package (which sometimes consists of more than one disk) contains an entire month's listings. If you want to order a disk package from a previous month, please call (603) 924-9281 to find out how many disks it includes. To order listings (for noncommercial use only), fill out this form and send a check or money order in the correct amount to:

BYTE Listings
One Phoenix Mill Lane
Peterborough, NH 03458

All prices include postage. Program listings can also be downloaded via BYTEnet Listings at (617) 861-9764.

BYTE issue: _____

COMMON 5¼-INCH FORMATS

All cost \$8.95, \$10.95 outside U.S.A. Annual subscription is \$69.95, \$89.95 outside U.S.A.

- ☐ Apple II 5¼-inch DOS 3.3
- ☐ Apple II 5¼-inch ProDOS
- ☐ Hewlett-Packard 125
- ☐ IBM PC
- ☐ Kaypro 2 CP/M
- ☐ Texas Instruments Professional
- ☐ TRS-80 Model III
- ☐ TRS-80 Model 4
- ☐ Zenith Z-100

COMMON 3½-INCH FORMATS

All cost \$9.95, \$11.95 outside U.S.A. Annual subscription is \$79.95, \$99.95 outside U.S.A.

- ☐ Apple Macintosh
- ☐ Atari 520ST
- ☐ Commodore Amiga
- ☐ Data General/One
- ☐ Hewlett-Packard 150

CP/M STANDARD 8-INCH FORMAT

All cost \$9.95, \$11.95 outside U.S.A. Annual subscription is \$79.95, \$99.95 outside U.S.A.

- ☐ Single-sided single-density
- ☐ Double-sided double-density

OTHER FORMATS

Due to the diversity of requests and the custom work involved, there will be some delay in fulfilling these requests. All cost \$9.95, \$11.95 outside U.S.A. Annual subscription is \$79.95, \$99.95 outside U.S.A.

Size ☐ 8-inch ☐ 5¼-inch ☐ 3½-inch
Machine _____

SEND TO:

Name _____

Street _____

City _____ State or Province _____

Postal Code _____ Country _____

Check or money order enclosed for \$ _____

BULLETIN BOARDS IN CANADA

Listed below are some computer bulletin boards that carry program listings from BYTE. Programs are for noncommercial use in connection with BYTE articles only. Some BBSs may charge an annual maintenance fee, and you must pay your own telephone charges.

Western Canadian Distribution Center (3420 48th St., Edmonton, Alberta T6L 3R5) will be supplying listings to its member bulletin board systems.

Edmonton, Alberta, (403) 454-6093

Meadowlark, Alberta, (403) 435-6579

Montreal, Quebec, PComm Systems, (514) 989-9450

Prince George, British Columbia (604) 562-9519

Regina, Saskatchewan, (306) 586-5585

Canadian Remote Systems, Toronto

Toronto, Ontario, Epson Club of Toronto (EPCOT), (416) 635-9600

Winnipeg, Manitoba, (204) 452-5529

In addition, arrangements for BYTEnet Listings have been made with one or more system operators in the following nations: Australia, Austria, Brazil, Denmark, France, Hong Kong, Indonesia, Italy, Japan, Malaysia, The Netherlands, Nigeria, Norway, Saudi Arabia, Singapore, Sweden, Switzerland, United Kingdom, and West Germany. Contact us at (603) 924-9281 for an up-to-date list. ■

EDITORIAL CALENDAR

1987

MAY — DESKTOP PUBLISHING: An exploration of the hardware and software needed for desktop publishing, from page description languages to high-resolution printers and typesetting back ends.

JUNE — COMPUTER-AIDED DESIGN: The anatomy of computer-aided design/drafting software, the graphics display devices needed for CAD, and the data structures used by CAD programs to export data to other applications.

JULY — LOCAL AREA NETWORKS: The technology of linking personal computers together to share data files, programs, and peripheral devices.

AUGUST — PROLOG: A look at logic programming with articles on tips and techniques and explorations of the tasks Prolog is best suited for.

SEPTEMBER — PRINTER TECHNOLOGIES: An examination of the state of the art in printer technologies, including laser, liquid-crystal shutter, and ink-jet technologies.

OCTOBER — HEURISTIC ALGORITHMS: Artificial intelligence techniques for giving computers the ability to learn from experience.

NOVEMBER — HIGH-PERFORMANCE WORKSTATIONS: A tour of the technology underlying the workstations used by scientists and engineers in computer-aided engineering/design.

DECEMBER — NATURAL LANGUAGE PROCESSING: The technology of getting computers to understand the natural language of man.

1988

JANUARY — MANAGING MEGABYTES: Looking at the ways computers store and retrieve data in situations where disk space is measured in gigabytes and memory is measured in megabytes. Also a look at the new applications that mega-memory and storage will permit.

FEBRUARY — LISP: A BYTE reexamination of the original language of artificial intelligence research.

MARCH — FLOATING-POINT PROCESSORS: A look at the processors that speed the computation of mathematical operations in personal computers, including coprocessors and array processors.

APRIL — MEMORY MANAGEMENT: The hardware and software issues in managing a personal computer's memory space.

MAY — CPU ARCHITECTURES: An exploration of the latest 32-bit microprocessors, including digital signal processors and programmable graphics processors.

Six great reasons to join BIX today

● Over 140 microcomputer-related conferences:

Join only those subjects that interest you and change selections at any time. Take part when it's convenient for you. Share information, opinions and ideas in focused discussions with other BIX users who share your interests. Easy commands and conference digests help you quickly locate important information.

● Monthly conference specials:

BIX specials connect you with invited experts in leading-edge topics—CD-ROM, MIDI, OS-9 and more. They're all part of your BIX membership.

● Microbytes daily:

Get up-to-the-minute industry news and new product information by joining Microbytes Daily and What's New Hardware and Software.

● Public domain software:

Yours for the downloading, including programs from BYTE articles and a growing library of PD listings.

● Electronic mail:

Exchange private messages with BYTE editors and authors and other BIX users.

● Vendor support:

A growing number of microcomputer manufacturers use BIX to answer your questions about their products and how to use them for peak performance.

What BIX Costs...How You Pay

ONE-TIME REGISTRATION FEE: \$25

Hourly Charges: (Your Time of Access)	Off-Peak 6PM-7AM Weekdays Plus Weekends & Holidays	Peak 7AM-6PM Weekdays
BIX	\$9	\$12
Tymnet*	\$2	\$6
TOTAL	\$11/hr.	\$18/hr.**

* Continental U.S. BIX is accessible via Tymnet from throughout the U.S. at charges much less than regular long distance. Call the BIX helpline number listed below for the Tymnet number near you or Tymnet at 1-800-336-0149

** User is billed for time on system (i.e., 1/2 Hr. Off-Peak w/Tymnet = \$5.50 charge.)

BIX and Tymnet charges billed by Visa or Mastercard only.

BIX HELPLINE

(8:30 AM-11:30 PM Eastern Weekdays)

U.S. (except NH)—1-800-227-BYTE

Elsewhere (603) 924-7681



We'll
Send
You a

BIX User's Manual and Subscriber Agreement
as Soon as We've Processed Your Registration.
JOIN THE EXCITING WORLD
OF BIX TODAY!

JOIN BIX RIGHT NOW:

Set your computer's telecommunications program for full duplex, 8-bit characters, even parity, 1 stop bit OR 7-bit characters, even parity, 1 stop using 300 or 1200 baud.

Call your local Tymnet number and respond as follows:

Tymnet Prompt

Garble or "terminal identifier"
login:
password:
mhis login:
BIX Logo—Name:

You Enter

a
byteneti <CR>
mgh <CR>
bix <CR>
new <CR>

After you register on-line, you're immediately taken to the BIX learn conference and can start using the system right away.

FOREIGN ACCESS:

To access BIX from foreign countries, you must have an account with your local Postal Telephone & Telegraph (PTT) company. From your PTT enter 310600157878. Then enter bix <CR> and new <CR> at the prompts. Call or write us for PTT contact information.

BIX

ONE PHOENIX MILL LANE
PETERBOROUGH, NH 03458
(603) 924-9281

Announcing BYTE's New Subscriber Benefits Program

Your BYTE subscription brings you a complete diet of the latest in microcomputer technology every 30 days. The kind of broad-based objective coverage you read in every issue. *In addition*, your subscription carries a wealth of other benefits. Check the check list:

DISCOUNTS

- ✓ 13 issues instead of 12 if you send payment with subscription order.
- ✓ One-year subscription at \$21 (50% off cover price).
- ✓ Two-year subscription at \$38.
- ✓ Three-year subscription at \$55.
- ✓ One-year GROUP subscription for ten or more at \$17.50 each. (Call or write for details.)

SERVICES

- ✓ **BIX:** BYTE's Information Exchange puts you on-line 24 hours a day with your peers via computer conferencing and electronic mail. All you need to sign up is a microcomputer, a modem, and telecomm software.
- ✓ **Reader Service:** For information on products advertised in BYTE, circle the numbers on the Reader Service card enclosed in each issue that correspond to the numbers for the advertisers you select. Drop it in the mail and we'll get your inquiries to the advertisers.
- ✓ **TIPS:** BYTE's Telephone Inquiry System is available to



subscribers who need *fast response*. After obtaining your Subscriber I.D. Card, dial TIPS and enter your inquiries. You'll save as much as ten days over the response to Reader Service cards.

- ✓ **Disks and Downloads:** Listings of programs that accompany BYTE articles are now available free on the BYTEnet bulletin board, and on disk or in quarterly printed supplements.
- ✓ **Microform:** BYTE is available in microform from University Microfilm International in the U.S. and Europe.
- ✓ **BYTE's BOMB:** BYTE's Ongoing Monitor Box is your direct line to the editor's desk. Each month, you can rate the articles via the Reader Service card. Your feedback helps us

keep up to date on your information needs.

- ✓ **Customer Service:** If you have a problem with, or a question about, your subscription, you may phone us during regular business hours (Eastern time) at our toll-free number: 800-258-5485. You can also use Customer Service to obtain back issues and editorial indexes.

BONUSES

- ✓ **Annual Separate Issues:** In addition to BYTE's 12 monthly issues, subscribers also receive our annual IBM PC issue free of charge, as well as any other annual issues BYTE may produce.
- ✓ **BYTE Deck:** Subscribers receive five BYTE postcard deck mailings each year—a direct response system for you to obtain information on advertised products through return mail.

To be on the leading edge of microcomputer technology *and* receive all the aforementioned benefits, make a career decision today. Call toll-free weekdays, 8:30am to 4:30pm Eastern time: 800-258-5485.

*And... welcome to
BYTE country!*

BYTE
THE SMALL SYSTEMS JOURNAL

